

INTRODUCTION

TABLE OF CONTENTS

SYSTEM CONFIGURATION	0-7
PANEL MONITOR (PAM-8)	0-10
CONSOLE DEBUGGER (BUG-8)	0-11
HEATH TEXT EDITOR (TED-8)	0-12
HEATH ASSEMBLY LANGUAGE (HASL-8)	0-14
BENTON HARBOR BASIC	0-15
TAPE FILES	0-17
System Record Structure	0-17
Label Record Format	0-18
System Data Formats	0-19
Reading the Displays	0-20
USING THE MAGNETIC TAPE SYSTEM	0-22
Recorder Operating Hints	0-22
PRODUCT INSTALLATION	0-25
Available Configuration Options	0-26
Generating a Configured Tape	0-27
Loading From a Configured Tape	0-29
Copying an Existing Memory Tape	0-29
Using an ASR Console	0-30
Using a 110-Baud Console Terminal	0-30
H8-4 Port Allocation	0-31
CONTROL CHARACTERS	0-32
CONSOLE DRIVER	0-33
REPORTING SOFTWARE PROBLEMS	0-34
APPENDIX A (Installing a Patch)	0-35
APPENDIX B (Console Driver Documentation)	0-37
APPENDIX C (I/O and Memory Maps)	0-48
APPENDIX D (ASCII Characters)	0-50
APPENDIX E (Number Conversion)	0-54
APPENDIX F (Memory Table, Offset Octal & Decimal Boundaries)	0-57
INDEX	0-58

**SOFTWARE REFERENCE
MANUAL**

CASSETTE SYSTEM

Model H8-18

for the
H8 Digital Computer System

Contents

NOTE: An individual Table of Contents is included at the beginning of each of the following sections.

Introduction	0-3
Panel Monitor (PAM-8)	0-10
Console Debugger (BUG-8)	0-11
Heath Text Editor (TED-8)	0-12
Heath Assembly Language (HASL-8)	0-14
BENTON HARBOR BASIC	0-15

SPECIAL DISCLAIMER

Heath cannot provide consultation on user-developed programs or modified versions of Heath Software products.

These software products were developed for the Heath Company by the Wintek Corporation. Software copyrights reside with Wintek Corporation.

This Software Reference Manual includes all the information you will need to become familiar with the software products supplied with your H8 Computer. These software products are: the Front Panel Monitor, PAM-8; the Console Debugger, BUG-8; the Heath Text Editor, TED-8; the Heath Assembly Language, HASL-8; and Benton Harbor BASIC, Heath Company's version of Dartmouth BASIC . . . NOTE: Extended Benton Harbor BASIC is available as an optional accessory.

This book is intended as a reference manual, and, as such, it is as complete as possible. Examples are included to help you understand exactly how the Heath software products carry out their instructions; but they are not designed to teach you programming. If you have never used a text editor and assembler, for example, we recommend that you obtain some instruction from other sources, such as the "Heathkit Continuing Education" courses, prior to reading this material. If you have used editors and assemblers, this Manual will tell you about the special features in the Heath Text Editor (TED-8) and the Heath Assembly Language (HASL-8).

This introduction describes each product briefly and covers those aspects of the package that are common to all. A separate section then follows for each software product. Each section provides detailed reference information and is followed by one or more Appendices for that product. Be sure to read all of this introductory section so you have a good overview of all the products.

Heath software products feature a high degree of commonality in many of the modules which make up the individual products. For example, all software products except for BASIC that use the console terminal employ an identical software module called the Console Terminal Driver. This common usage of the console terminal driver permits you to move easily from one software product to the other, as the operating features are similar. Likewise, all tape handling is carried out through a common tape handling package, and once these features are understood, they are applicable to all products.

Heath software is supplied in two forms: cassette magnetic tape and read/only memory (ROM). The Panel Monitor (PAM-8) is supplied in a ROM (programs supplied in ROM cannot be modified by the user). The Console Debugger (BUG-8), the Heath Text Editor (TED-8), the Heath Assembly Language (HASL-8), and BASIC are supplied with the H8 in cassette form. The cassette tapes are compatible with the required error checking and synchronizing characters used by the front panel monitor system.

A printed copy of the panel monitor source listing is provided to aid you in using PAM-8. The Console Driver Entry Points and the entry points of the BASIC floating point package are also included. All other programs are supplied in binary object forms and listings are **not** available.

SYSTEM CONFIGURATION

The following paragraphs will describe the minimum requirements for your operating system. It is impossible to list all the different combinations that you may want to use. But this information will let you configure the operating system so you can begin using your H8 computer right away.

This section only contains the information about the software configuration. Refer to the "System Configuration" section of your Assembly Manual for instructions on how to implement the hardware.

Obviously, you must have an H8 computer system with a minimum 12K bytes of memory. Moreover, this memory must operate continuously from decimal location 8192 (040.000A) to the available upper memory limit of your computer. Note that the lower 8191 bytes of memory are reserved.

The H8-5 Serial I/O and Cassette Interface Card (interface circuit board) provides a cassette interface that lets you load and dump binary data. The firmware in PAM-8 controls the operation of the cassette recorder when binary data is being transferred, and it directs the data flow through port addresses 370 and 371. The cassette interface operates at either 300 or 1200 baud and uses 2400 and 1200 Hz tones, respectively, for a logic 1 and logic 0 when transmitting data.

Your H8 microcomputer must be interfaced to a console terminal, such as the Heath H19, when you want to use a software product, such as TED-8, HASL-8 or BASIC. You may interface this console with either an H8-4 or H-5 Serial I/O card. These cards let you communicate with a console at standard baud rates (110, 150, 300, 600, . . . , 9600). The serial communication should be wired for RS-232 usage.

This Serial I/O card provides the interface between your H8 microprocessor and the peripheral device, such as a console terminal or line printer. You must select and jumper the card, to enable the correct port address, before using the software. The system will operate only when the hardware and software configuration agree.

Some circuit boards use a fixed interrupt and port allocation scheme and cannot be adjusted. For example, the controller board on the floppy disk hardware. This information is important when you want to interface other peripherals.

Table 0-1 outlines the standard port allocation scheme used on the H8 computer system, and lists the name of each device. The console terminal for example, is referenced by the software as TT: and the line printer as LP:. The name of the interface card and the Port address are listed after the device name.

Be sure that you have installed a jumper on the interface card that corresponds to the correct interrupt vector. (Refer to the "System Configuration" section of your Assembly Manual.) For example, a console connected to the H8 with either an H8-4 or an H8-5 requires that you select interrupt vector three.

The H8 operating system supports any standard baud rate up to 9600 baud. However, the device and the interface card must be using the same baud rate. This rate is hardware selectable on all cards except the H8-4, which is software programmable and lets you select a baud rate for each channel.

The H8 operating system does not and will not use any port address below 100 octal. These ports are reserved for your personal use.

TABLE 0-1

<u>DEVICE</u>	<u>DEVICE NAME</u>	<u>INTERFACE CARD</u>	<u>PORT ADDRESS</u>
Reserved			376 - 377
Console Terminal	TT:	H8-5 Serial	372 - 373
		H8-4 Serial	350 - 357
Cassette	--*	H8-5 Serial	370 - 371
Line Printer	LP:	H8-4 Serial	340 - 347
Aux. Terminal	AT:	H8-5 Serial	374 - 375
		H8-4 Serial	300 - 307
Front Panel	--*	--*	360 - 361
Floppy Disk	SY:	Special	174 - 177
Reserved			170 - 173

* - None is used.

NOTE: Support for the H8-4 Multiport Serial I/O Card is only available in those versions numbers XX.03.XX or later. The XX value will vary for each software product. The line printer must be a (W)H14 at Port Address 340-341.

PANEL MONITOR (PAM-8)

The ROM Panel Monitor, which is permanently located in the lower 1024 bytes of memory, permits you to load, execute, and debug programs written in 8080 machine language. The Heath Panel Monitor also makes use of the first 64 locations of random access memory. The H8 front panel is used as an I/O device, and it is assigned port numbers 360 and 361. With the Heath Panel Monitor, you can:

1. Examine the contents of a memory location.
2. Change the contents of a memory location (enter a new program, for example, or modify an old program).
3. Examine the contents of any of the 8080 registers.
4. Change the contents of any of the 8080 registers.
5. Start or stop the execution of a user-written program.
6. Execute a user program, a single instruction at a time.
7. Dump a program onto magnetic tape, with error detection codes and synchronization data.
8. Load a program from magnetic tape into the desired memory locations.
9. Breakpoint a user program.
10. Reinitialize to a power up status.

The Heath Panel Monitor also offers the following features:

1. The user may automatically increment or decrement memory addresses which are being examined or modified.
2. The user may automatically increment or decrement through the registers which are being examined or modified.
3. The user is provided with a visual indication of the current mode in which the panel monitor is operating.
4. The user is provided with audio feedback upon valid and invalid command and data entry.
5. The H8 front panel utilizes an octal display rather than the more difficult to read binary display.
6. The front panel key switches and display are available for your programs.
7. The front panel display is operated on a continuously updated basis and, therefore, is active even during the execution of a user program. This feature permits the user to monitor either registers or memory location while his program is operating.

PAM-8 provides the fundamental tape routines by which you load all other programs, including the Heath-supplied software.

CONSOLE DEBUGGER (BUG-8)

BUG-8 allows you to perform very sophisticated operations from a console terminal with a full active keyboard and display. BUG-8 resides in H8 memory, using approximately 3,000 bytes of storage. You can use BUG-8 to write, load, execute, and debug machine language programs in the H8 computer in octal, decimal, or ASCII format. This package also has many of the features included in PAM-8.

With the Heath Console Debugger, you can:

1. Examine the contents of memory locations.
2. Alter the contents of memory locations.
3. Examine the contents of the CPU registers.
4. Alter the contents of the CPU registers.
5. Start program execution.
6. Execute a program in a single step form.
7. Set break points with multiple hit capability.
8. Clear break points.
9. Load programs from magnetic tape.
10. Dump programs onto magnetic tape.

BUG-8 is an advanced monitor, permitting you to prepare extensive software in machine code format that can be readily debugged and then recorded on a mass storage unit for future use.

HEATH TEXT EDITOR (TED-8)

The Heath TED-8 Text Editor is a general purpose, line-oriented text editor that is used primarily to prepare source code that can be assembled using the Heath assembler (HASL-8). But while this is its primary purpose, TED-8 is also useful for such things as letter writing, preparation of club newspapers, and manuscript editing.

This software product requires an H8 system with 8192 bytes of memory, an ASCII keyboard for text entry, and an ASCII display for text display. If large files are to be used or files are to be saved, a separate input/output tape unit is recommended.

With the Heath Text Editor, you can:

1. Read text from a pre-existing text file.
2. Create text for a new file.
3. Output text to a named tape file.
4. Insert new text after a given line.
5. Search the text for a given character string.
6. Delete a given line or lines.
7. Print a particular line or lines.
8. Replace a given line.
9. Edit a given string; that is, replace a particular string with another string.

All the above functions are supported by a number of special features, some of which are only available on the Heath Text Editor. Some of these features are:

1. A wide scope of range expressions, including:
 - A. First line.
 - B. Last line.
 - C. Single line.
 - D. Line to line.
2. Count and string versions of range expressions, which permit you to edit lines, plus or minus a certain number of lines from a given line, or to edit all lines containing a certain string.
3. You have the option of selecting one of three optional modes. Optional mode A prints the line after operating on it, optional mode B prints the line before operating on it, and optional mode BA prints the line before and after operating on it.
4. The use of a Qualifier String (Qualifier Strings permit operating only on the lines containing designated strings).
5. Tab. This command lets you set tab stops for entering text. The editor is constructed so that tabs do not occupy extensive user storage.
6. A Use statement, which provides a line count and memory usage information.
7. File Labeling Procedures to create new file names in either the input or output mode.

Under the H8 text editor, source code is prepared for the Heath Assembly Language (HASL-8). Once the source code has been prepared, it is written to a cassette tape or paper tape output file. Once this has been done, the user proceeds to the assembler.

HEATH ASSEMBLY LANGUAGE (HASL-8)

Heath Assembly Language runs on a Heath H8 Computer using about 8192 bytes of memory. This program assembles source code and produces object code. HASL-8 utilizes all the standard 8080 mnemonics, extended mnemonics, and numerous psuedo instructions.

Some of the special features of HASL-8 are that it:

1. Recognizes five operators: plus, minus, *, /, unary-.
2. Recognizes four token operand expressions:
 - A. Integers.
 - B. Symbols.
 - C. Character strings.
 - D. The origin symbol.

HASL-8 is a two pass assembler. Before the user starts assembly, it asks if a binary output is to be generated. On the second pass, it produces the binary if directed to do so, as well as the appropriate listing. The binary object code may be placed on a specified output device or may be placed directly in memory.

HASL-8 features the same terminal controls as do other Heath programs, including a suspend output mode and a discard output mode.

BENTON HARBOR BASIC

BENTON HARBOR BASIC is a modified version of Dartmouth BASIC, an easy-to-learn-and-use conversational language.

The BENTON HARBOR BASIC system is interpretive. That is, it executes each statement as it comes to it. BENTON HARBOR BASIC uses an H8 computer with 8K of memory, an appropriate terminal, and magnetic tape handling capability. Some of the features of BENTON HARBOR BASIC are:

1. Two different data types:
 - A. Numeric data, which has over six digits of accuracy and lies in the range of 10^{-39} to 10^{+38} . Numeric data may be either fixed or floating point.
 - B. Boolean values, which permit logical operations.
2. Multidimensioned variables.
3. BASIC supports fifteen operators, which are:
 - A. \neg (unary) NOT
 - B. \uparrow Exponentiation
 - C. $*$ /
 - D. $+$ $-$
 - E. $<$, $<=$, $=$, $<>$, $>=$, and $>$
 - F. OR
 - G. AND

4. Free Format Programs.
5. Multiple statements per line.
6. Enhanced expression and conditional statement facilities.

BASIC features both command and program modes, where statements may be executed immediately after the line is written or numbered lines may be used so the program will not be executed until a RUN statement is executed.

BASIC also features command completion. In the command mode, BASIC checks inputted characters and, as soon as there are sufficient characters to establish a unique command, the command is completed. This feature saves considerable typing time and reduces errors.

NOTE: In order to fully use the Heath Software package, you must not only review the special features of Heath programs, but you must also know how to use monitors, debuggers, Text Editors, Assemblers, and BASIC. Once you have learned to use such programs, this Software Reference Manual will be an invaluable quick reference on how to carry out specific functions within the H8 software packages.

TAPE FILES

This section describes the tape format used in the Heath H8 Computer System. Tape formats are identical, regardless of the media used. The following terms are used to define the Heath H8 Tape format.

FILE A logically complete set of data. For example, a memory dump causes the FILE to be written on the tape. Although several files may be written onto one tape, the files are each totally independant of any other information written on that tape. A file consists of one or more records.

RECORD A record is a discrete block of data written to the tape transport. Each record must be read all at one time. It is not possible to read part of the record, pause, and then read the rest. Each record contains a CRC-16 Check. Each file has a first and last record. They may be the same record in a one-record file. The records in the file are numbered so a missing record can be detected.

System Record Structure

As previously discussed, all H8 files consist of one or more records. All of the records have the same format.

SYNs	STX	E O F	TYPE	SEQ	COUNT	Data . . . Data	CRC-16
------	-----	-------------	------	-----	-------	-----------------	--------

SYN From 20 to 40 ASCII Synchronizing Idle (026) characters.

STX An ASCII STX character. This character, preceded by at least 10 SYN characters, indicates the start of a record. The SYN characters and the STX character are not included in the CRC. Note that a gap may be required between records to allow the tape transport to start and stop.

EOF End of file. This flag is the high-order bit in the 'TYPE' byte. If set, it indicates that this is the last record in the file. The record is otherwise normal, and may contain data.



- TYPE This 7-bit field (the 8th bit is 'EOF') indicates the type of the record. All records in a file have the same type. The data field's format is type dependent. See below for a description of file types.
- SEQ This field is an 8-bit sequence counter, used to detect missing records. If a label record is present in the file, it is record #0. The first data record is #1. If the file contains no label record, the first record is record #1. Note that the record following record #255 is record #0, but is not a label record.
- COUNT This two-byte field contains a count of the number of bytes in the Data field. The high-order byte of the count appears first. Note that the count may be zero, indicating that there is no data field.
- DATA This field contains the data. Its format is dependent upon the record type. Its length is set in 'count'.
- CRC-16 This is a polynomial remainder check, computed byte-wise upon the entire record (starting with the EOF/TYPE byte) from $(X + 1) * (X^{15} + X + 1)$
This checksum provides nearly flawless error detection.

<u>ERROR</u>	<u>DETECTION RATE</u>
Single bit error	100%
Double bit errors	100%
An odd number of bits in error	100%
An error burst <17 bits long	100%
An error burst >= 17 bits	99.997%

Label Record Format

Some file types require a label record to be present, and some require that no label record be present. A label record is detected by its record number of 0. Except for the contents of the data field, a label record has the same format as the other records in the file. The data field consists of a string of 7-bit ASCII characters which comprise the file's label. The 8th bit should be 0 for all characters.



System Data Formats

The following section describes the data formats associated with the various file types. There are currently three file types:

1. Memory Image.
2. BASIC programs.
3. Compressed Text.

MEMORY IMAGE (Type = 001)

The file type 'memory image' is used when you dump or load programs from H8 memory. This file type has no label record, the first record in the file is #1. The file may consist of one or more records. The format of the data field is:

ENTRY	ADDR	Program bytes
-------	------	---------------

Where ENTRY = the program's entry point address, and ADDR = the address to start loading this group of program bytes. If there are multiple records in this data file, the 'entry' portion of each record should be identical.

NOTE: The COUNT field in the record header does not include the 4 bytes for ENTRY and ADDR. Thus, an empty record of this type has a zero COUNT field, but still contains the ENTRY and ADDR in the data field. Note that the high-order byte comes first for the ENTRY and ADDR fields.

BASIC PROGRAM (Type = 002)

This file type is used by BASIC when you load and dump programs. The file always has a label record (#0), and always has only one data record, #1. The data field contains the BASIC program in a special internal format. This file type can not be processed by the text editor.

COMPRESSED TEXT (Type = 003)

This file type is used by TED-8 and HASL-8 for source statements. It always has a label record (record #0) and has one or more data records. The data field in each record should not exceed 512 bytes. Lines should not be split between records. Each line is compressed according to the following format:

1. All characters are 7-bit ASCII, with the parity bit zero.
2. The carriage return and line-feed characters are not used. The end of line is indicated by a 000 byte.

3. Strings of spaces are represented by the value $200_8 + N$, where 'N' is the number of spaces in the series. Thus, a single blank is encoded as 201_8 ; ten blanks are encoded as 212_8 .
4. The maximum line length is 127 characters.

Reading the Displays

When the H8 computer is reading or writing data on a tape transport, the front panel displays are continually displaying data about the tape operation. Data about tape operation is displayed into two areas:

ADDRESS LEDs — Display the number of bytes left in the record when the transport is reading or writing data. The Address LEDs do not display any information during the inter-record gap. The address LEDs display the actual address being loaded when a memory image load or dump is executed. During a memory image operation, the Data LEDs display the data being entered into or read from memory.

DATA LEDs — Display the type of data and the record number. The right-hand-most data LED displays the type of data being read or written. This information is displayed as:

<u>DISPLAY</u>	<u>DATA/TYPE</u>
1.	Memory Image
2.	A BASIC program.
3.	Compressed text.

The two left hand LEDs display the octal record count within the particular file. As noted earlier, a file may contain one or more records. When 17_8 records are exceeded, the record count in the two left hand data LEDs starts over at 00. When the last record is read, the extreme left hand data LED displays a two or a three if the record count is between 10_8 and 17_8 . Using this information, you can readily observe the type of data being handled by the H8. Figure 0-1 shows how these displays are used.

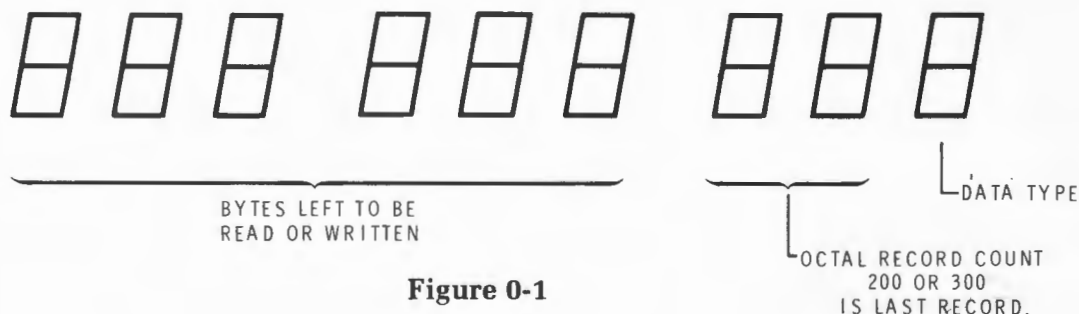


Figure 0-1

For example:

2 1 3

indicates the first and last data record of compressed text. NOTE: This could be the 1st or the 17₈th data record.

0 1 1

indicates first data record of a memory image file.

0 0 3

indicates a label record of a compressed text file. Note, the label record is record number 0.

2 3 3

indicates the third and last record of a compressed text file.

3 1 1

indicates the last record of a memory image file.

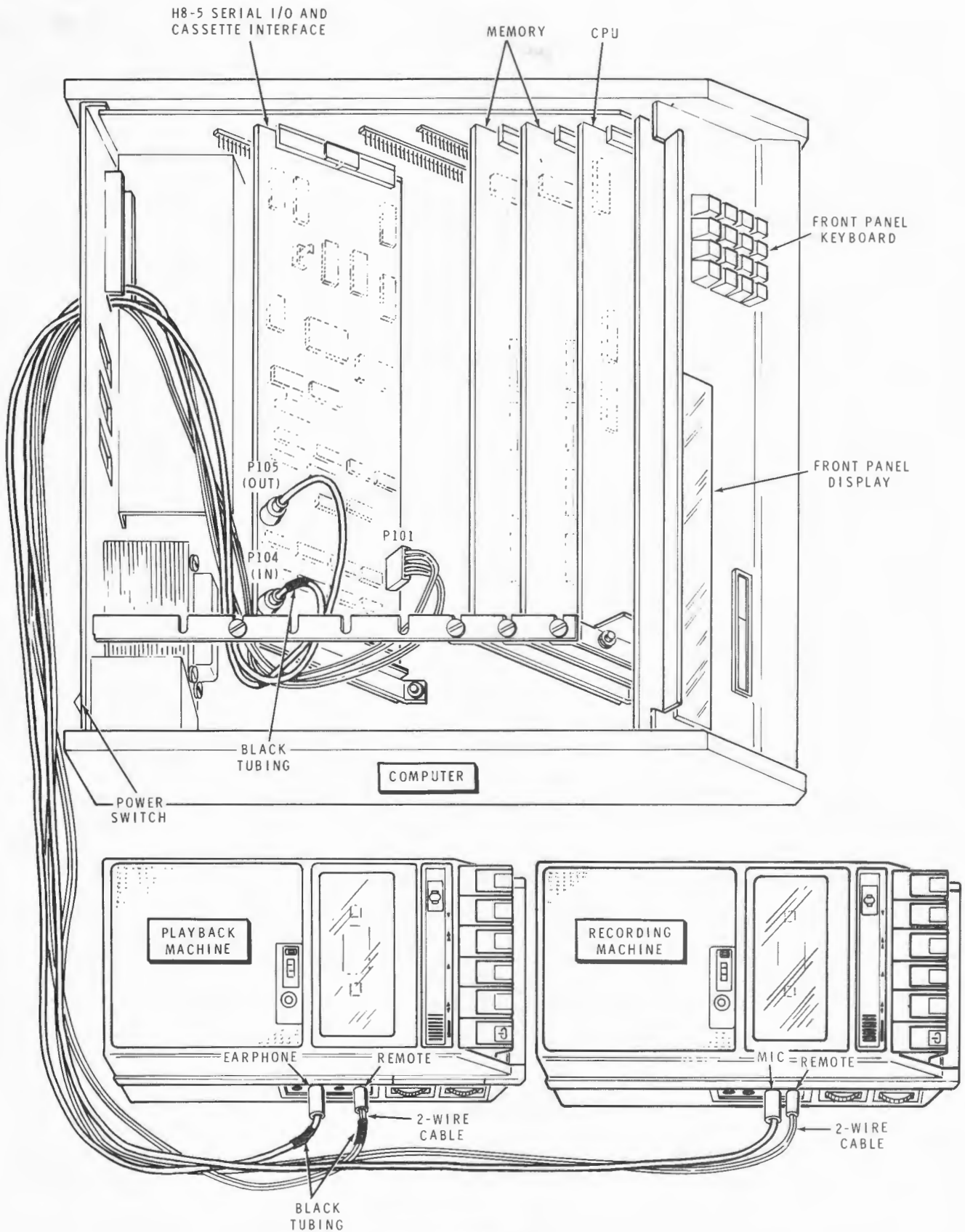
USING THE MAGNETIC TAPE SYSTEM

The Heath H8-5 serial card supports two different configurations of magnetic tape recorders. You may use a single tape recorder or a dual tape recorder. See Pictorials 1A and 1B. (The H8-5 assembly manual describes the jumper wires used.) The most versatile system operation is obtained by using two tape recorders with independent control. However, you can achieve a perfectly workable and somewhat less expensive system with the single recorder. Dual recorders are preferable when you are assembling long programs, as it is necessary to read a few records of the source program and then assemble this source material, generating the appropriate binary output before reading additional source records. If you use a single recorder, you must change cassettes frequently.

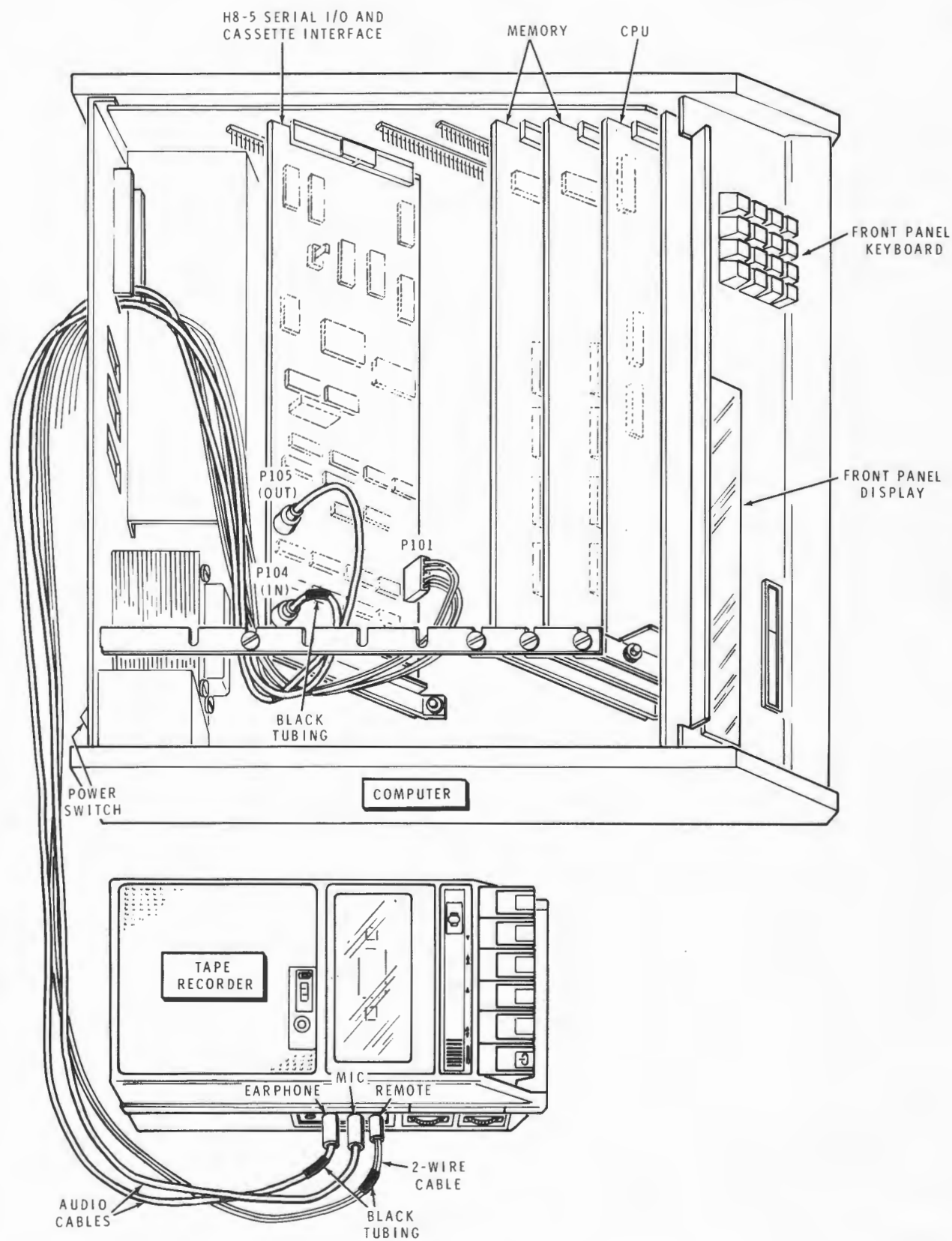
Recorder Operating Hints

Observe the following guidelines carefully to get the maximum operating efficiency from your H8 Cassette Recorder system.

1. Use only Heath approved cassette tape recorders. Although there are a variety of very good tape recorders on the market, only those models tested and approved by the Heath Company will assure successful operation.
2. Use a high-quality cassette recording tape. Once again, the Heath approved tape is required to assure success.
3. Be sure the tape is off the leader before recording a file. Frequently cassettes have excessively long leaders. Therefore, the initial portion of a file placed on tape may lose the synchronizing characters, and be lost.
4. Keep the tape and the recorder clean. Dirty tapes tend to cause drop outs which cause tape errors.
5. Label your tapes. There is nothing more frustrating than a good program written onto a tape that is unlabeled and therefore not recoverable later.



PICTORIAL 1-A



PICTORIAL 1-B

PRODUCT INSTALLATION

To install a software product (BUG-8, for example), you should create a configured tape from the distribution tape originally supplied with your H8 computer system. The distribution tape should then be safely stored away and you use the configured tape or tapes when you operate your system. This procedure lets you "personalize" and protect your H8 software. Remember, a major part of your H8 computer system is the software, and creating this configured tape protects your investment.

Use the following configuration procedure when you initially install the tape operating system. In addition, when you make changes to your H8 operating system, you must consider the possibility of modifying or updating your configured tapes. For instance, assume that the HIGH MEMORY limit was initially configured for 16K and several months later you purchase an additional 4K memory circuit board. The software on a configured tape will not recognize the new memory until you reconfigure a new tape from the distribution tape.

Place the distribution tape into the cassette and load the software product (i.e. BASIC, BUG-8, TED-8, or HASL-8) into the H8 computer memory using the normal LOAD procedure. Use the Panel Monitor to insert any optional software patches (See "Appendix A", Page 0-35). For instance, you must always insert a patch when operating the console at 110 baud.

Press the **GO** key on the H8 keyboard when you have installed all the patches. Repeatedly press the space bar on the console until an introductory message is printed; then select the configuration options available. Use the SAVE option and a new tape to create the "configured" tape. Store the distribution tape in a secure location away from heat, humidity, or magnetic fields.

Available Configuration Options

You must select the options when you configure your H8 software tape. You do not need to make any change when using Heath peripheral devices. Sometimes, your personal preference is involved in the choice. For instance, some programmers prefer to use a CTRL-U rather than a RUBOUT key to delete a line of text. However, special applications will require that you make the appropriate change. To make a change, simply prompt the H8 by typing the first character on the console keyboard. The options are:

AUTO NEWLINE (Y/N)? — A yes (Y) response to this question directs the product to generate a new line each time the print head (or cursor) moves out of the last column of the console terminal. This function is distributed preset to Y.

BKSP=00008/ — The backspace character is normally a control H (00008 decimal). When used with the video terminal or other backspacing devices, the control H generates a true backspace. You may change the backspace character to other ASCII printing or nonprinting characters (see "Appendix D"), such as a backslash, if you are using a non-backspacing terminal. This new character will be considered a true backspace by the software.

CONSOLELENGTH=00080/ — The console length is initially set at 80 (decimal) characters, the normal width of a video terminal. You may change this to other common values, such as 132 characters for a wide printing terminal, or to 72 characters for a teleprinter.

NOTE: The maximum number of characters per line is a function of each software product. See the individual sections to determine the maximum permissible characters per line.

HIGH MEMORY=XXXXX/ — When the software product is initially started, the limit of available high memory is determined. All products start at 040 100 (offset octal). If you wish not to use a certain portion of high memory, type in a new high memory limit (decimal count). If the upper memory limit is set too low, the new limit will be refused (the terminal bell will sound).

LOWER CASE (Y/Y)? — A Yes (Y) response to LOWER CASE configures the software product to input lower case letters and output lower case letters. A N (no) response to the question configures the product to work with upper case only terminals. This function is distributed preset to N.

FLAG DEMO
FLAG, DEMO
FLAG
FLAG PROG
F BUG
FLAG BUG

PAD=4/ — The pad characters (nulls) are inserted following a carriage return. The pad characters are sent at this time to allow the print head time to return to the left-hand margin. For video terminals, and most teleprinters, the number of pad characters may be changed to zero. If you do not know how many pad characters are required for your terminal, initially try zero. If you appear to be overtyping (or missing characters) at the beginning of lines, increase the pad count until the overtyping stops. You may enter up to a maximum of 9 pad characters.

RUB OUT=00127/ — The rub out character deletes a line of text. If you desire to use a special rub out character, for instance, CTRL-U you must change it by entering a new decimal number (21) identifying a different ASCII character. See "Appendix D."

SAVE? — A yes (Y) in response to this question directs the software product to generate a memory image of the configured product. This memory image of the configured product should be the tape you use regularly to load your program. This will avoid your having to configure the product on a regular basis. Before executing the save command, be sure the tape transport at the dump output is ready.

To use the product directly from the distribution tape, type the return key at any time rather than typing a key which prompts a question.

NOTE: It is very important that you immediately configure products as you use them, and then place your original software distribution tape in an appropriate place for safe keeping. Use the above procedure any time you wish to configure the product.

Generating a Configured Tape

The Heath H8 software is supplied on distribution tapes. This software consists of a console debugger (BUG-8), text editor (TED-8), assembler (HASL-8), and an interpreter (BASIC). Use the following procedure to generate a backup tape from the software distribution tape:

1. Initiate PAM-8 by applying power to the H8 computer system. If the system is operational, you can reset the Panel Monitor by simultaneously pressing the 0 and RST/0 keys on the H8.

2. Install and rewind the distribution tape. (The H8 will activate the cassette motor when you press the LOAD key.)
3. Press the PLAY button on your recorder and, if necessary, the LOAD key on the H8. The H8 signals a successful LOAD by sounding a single beep. (A series of beeps indicates a tape error and you will have to repeat steps 1, 2 and 3.)
4. Use PAM-8 to install any optional patches. (See "Appendix A".)
5. Check the operational status of the console terminal.
6. Press GO on the H8 front panel.
7. Press the space bar on your console terminal several times until an introductory message is printed.

NOTE: Because the H8 computer system recognizes multiple peripheral devices, you must signal the location of your console terminal by pressing the space bar.

Repeat steps one through seven when you wish to LOAD any Heath software distribution tape.

You must now decide what options, if any, to use with your computer system. Configure the "new" software product as desired.

Because the objective was to create a configured tape, replace the distribution tape with a blank cassette. Use the recorder and the SAVE option to copy a duplicate tape. This new tape becomes your "CONFIGURED TAPE" and the distribution tape should be safely stored. You can repeat the SAVE option to create additional configured tapes.

When you are satisfied with the selected options, press the return key on your console terminal keyboard. The software product is ready to use and it will announce itself by printing a header message on your terminal.

Loading From a Configured Tape

Loading from a configured tape is a very simple procedure. It is the recommended way to normally load the software. The procedure is:

1. Load the tape in the tape transport.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. Wait for a single beep, indicating a successful load.
5. Press the GO key on the H8 front panel.
6. Press the space bar on your console terminal several times.
7. The console terminal will respond with the product description and its prompt character. The product is now ready to use.

Copying an Existing Memory Tape

Use the following procedure to copy a memory image tape. Be sure to use this procedure. Memory image tapes should not be copied on an audio-to-audio basis. An audio-to-audio copy may not work. To copy a tape:

1. Load the source tape in the tape transport.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. Wait for a single beep, indicating a successful load.
5. Load a blank tape into the dump tape transport.
6. Ready the dump tape transport.
7. Press DUMP on the H8 front panel.
8. Wait for a single beep, indicating a completed dump.
9. Repeat steps 7 and 8 to produce a second dump, creating a double copy.

The product is now copied and ready to be used. **IMPORTANT:** Make at least one double copy of the distribution tapes you received with the H8 as a protection against accidental tape damage. Once the tape is copied, if you are using magnetic tape, knock the read-only plugs out of the back of the cassette.

Using an ASR Console*

The following procedure allows you to use an ASR console as the main load/dump port, as well as the console terminal with an H8 system. An example of such an ASR (automatic send/receive) console would be the Teletype Corporation Model 33 Teleprinter. Perform the initial load by first setting the port interchange switch to the port interchange position on your H8-5 Serial I/O card. The tapes are then read in, in accordance with the procedure outlined under "Creating a Configured Tape." Once the tapes are read in, PAM-8 should be used to patch the software to the ASR console terminal configuration.

The appropriate patches for the ASR console terminal configurations of the software products are found in Appendix A of this section. Once the patches are accomplished, the GO key may be pressed and the normal configuration procedure takes place. Leave the Port Interchange switch in the Port Interchange position as long as the tape handler on the console terminal is used as the main load dump terminal.

Using a 110-Baud Console Terminal*

If you use a 110-baud console terminal such as a teleprinter, one extra stop bit must be added to the ASCII characters sent by the H8. This change is made at configuration time and should be done any time you use a 110-baud console, regardless of whether or not you use the terminal as an ASR console (such as a teleprinter with a paper tape reader/punch) or simply as the console terminal. The patch is listed in Appendix A.

*NOTE: If your system device uses an H8-4 Multiport Serial I/O Card, you need not make any patches for two stop bits. The H8-4 automatically supplies the stop bits when it encounters 110 baud.

H8-4 Port Allocation

The Heath H8-4 Multiport Serial I/O Card is a four-channel asynchronous interface. Compatibility with an H8 computer system can only be assured when the H8-4 port address allocation is as follows:

340Q	Line printer.
350Q	Console control terminal.

The H8-4 is supported by a specially designed software package. This software lets you operate with either or both of the H8-4 and the H8-5 Serial I/O Cards. When both of these circuit boards are being used, the console connected at port location 350Q becomes the controlling terminal.

Each channel on the H8-4 is fully programmable, including the baud rate, and uses its own input and output ports. Any channel is functionally independent of the other three and can be interfaced with any device using the standard RS232C configuration. In addition, channel zero (0) gives you a 20 mA current loop option.

CONTROL CHARACTERS

The characters Control-A, -B, -C, and -D are available for use within the program. For example, most Heath programs use Control-C as a general purpose cancel. The other characters are permanently assigned in the console driver, and therefore all software products using the console driver. These characters are assigned as follows:

CONTROL-O

Control-O toggles the output discard flag. When the output discard flag is set, output to the console terminal is stopped, but program execution continues. Typing Control-O once sets the discard flag. Typing the Control-O again resets the output flag, permitting output to the console terminal to resume.

CONTROL-P

Control-P resets the output discard flag. Typing Control-P insures the output discard flag is not set. NOTE: Control-O toggles the flag, but Control-P only resets it.

CONTROL-S

Control-S sets the suspend output flag.

CONTROL-Q

Control-Q resets the suspend output flag.

The above control characters are not echoed to the console terminal when they are typed as is a normal character. NOTE: Many Heath Software products also use Control-H and Control-I. These characters are not used by the console driver but are passed directly through to the program for individual processing. They are also echoed to the console terminal.

CONSOLE DRIVER

The console driver also provides capabilities for communicating with any I/O devices, including the console terminal and a tape transport at the load/dump ports. If you, the user, develop any of your own software packages, we recommend that you incorporate this console driver rather than attempting to develop your own console driver. (See "Appendix B".)

The console driver also provides two front panel entry points. These entry points are assigned as follows:

<u>PROGRAM COUNTER*</u>	<u>ENTRY TYPE</u>
040 100	Program Reset entry point. All text buffers, etc., are effectively cleared and the product is re-started. This is known as a "cold" or "hard" start.
040 103	Program restart entry point. Product is restarted with text buffers, etc., intact. This is known as a "warm" or "soft" start.

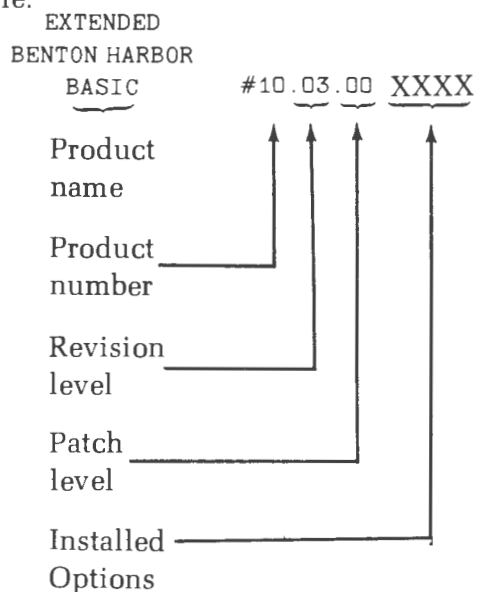
*NOTE: Set the value of the program counter using the front panel monitor. Then press GO.

REPORTING SOFTWARE PROBLEMS

Every effort has been made to keep the Heath H8 Software products free of defects. Should you suspect that a Heath H8 system software product may be defective, review the following procedure before contacting Heath Company.

1. Attempt to reload all software to be sure it has not been damaged.
2. Reconfigure the product from the distribution tape in an attempt to duplicate the problem.
3. If the problem persists, document the apparent product problem and the software version code. NOTE: It is extremely important that you know your exact software version code when you contact the Heath Company about your software product.

For example:



NOTE: Heath Company can not consult on user-developed programs or modified versions of the Heath Software products.

APPENDIX A

Installing a Patch

A patch is a modification to a software product. It is necessary to modify the software when a "BUG" is detected and corrected. The H8 console driver also requires that you install a patch when using either an ASR terminal or outputting data at 110 baud.

This Appendix shows you how to install a patch in BUG-8 that lets you interface your computer system to an ASR console terminal. When this patch is installed, all devices connected to your H8 computer system with an H8-5 circuit board will be supplied two stop bits.

NOTE: This patch is not required when your console is interfaced using an H8-4 Multiport Serial I/O Card.

To install a patch, insert and rewind the distribution tape in your cassette. Prepare the tape transport by pressing the PLAY button. Use the H8 front panel to LOAD the distribution tape. use the procedure outlined in PAM-8 page (1-1) to alter a memory location(s) and install the patch. Select the options for tape configuration. For instance, to install a patch in BUG-8:

1. Rewind and install the BUG-8 software distribution tape into your recorder. Press the PLAY button on your recorder.
2. Press the LOAD key on the H8 front panel.
3. Install the patch:
 - A. Press the MEM key on your H8 front panel and enter the address (040.220) of the data to be altered.
 - B. Press the ALTER key and enter the data (200).
 - C. Press the ALTER key a second time when you wish to exit the alter mode.

4. Press the GO key on the H8 front panel and select the desired configuration options. (See Available Configuration Options Page 0-26).

You can use an ASR terminal, an H8-5 Serial I/O and Cassette Interface Card and BUG-8 with the configured tape. Simply load the configured tape and follow the procedure in the BUG-8 Appendix for loading a configured tape ("Appendix A", Page 2-20). You will not have to install a patch every time you use the configured tape.

STORE THE DISTRIBUTION TAPE IN A SAFE LOCATION.

APPENDIX B

Console Driver Documentation

Machine language programs can use the H8 console driver. This approach increases your programs efficiency and lets you retain the control characters (i.e. CTRL-C, CTRL-S, etc.). This appendix lists the starting address of six frequently used console driver subroutine entry points that a machine language programmer might use.

We recommend that you use the BUG-8 console driver when writing machine language programs. An accepted procedure for combining the console driver and your program requires that you originate your program above the driver (ORG 44125A). You must also equate selected console driver entry points; for example, the console write character subroutine (\$WCHAR EQU 40111A).

Use HASL-8 to assemble your program, and generate a binary tape. This binary tape will be loaded above the console driver and you can combine the two programs with a panel monitor DUMP. For instance, assume your program was assembled and a binary tape was generated. Use the following procedure to combine the console driver and your program.

1. LOAD the console debugger (BUG-8) into H8 memory.
2. Reset PAM-8 and LOAD your binary tape over BUG-8. (NOTE: the console driver remains in H8 memory.)
3. Dump the combined program to tape starting from address 040.100 to the end of your program or the end of memory. Use the Panel Monitor dump routine (Page 1-18).

The console driver and your program are ready for execution. Simply reset the panel monitor and LOAD the new tape. PAM-8 signals a successful load by sounding a short beep. Press the GO key on the H8 front panel to execute your program.

...CONSOLE DRIVER DOCUMENTATION - HX-05.XX SOFTWARE
SYSTEM I/O DRIVER

HEATH H9450.V1A4...01/20/78
18.23:04 11-06N-79

PAGE 1

```

040.108      4      ORG      040108A
5      ***      CONSL1 - SYSTEM CONSOLE AND I/O DRIVER.
6      *
7      *      CONSL1 IS A GENERAL PURPOSE CONSOLE DRIVER. IT IS A STANDARD PRO-
8      *      DUCT USED IN ALL HEATH H8 SOFTWARE (WHICH COMMUNICATES WITH A CON-
9      *      SOLE DEVICE).
10     *
11     *      CONSL1 CONTAINS:
12     *
13     *      1. PORT ROUTINES. THESE ARE PLACED IN CONSOLE SO THEY HAVE
14     *      THE SAME LOCATION IN ALL PRODUCTS. PORT ADDRESSES MAY BE
15     *      CHANGED BY PATCHING THESE ROUTINES.
16     *
17     *      2. THE CONSOLE DRIVER PACKAGE. THIS PACKAGE CONSISTS OF SIX
18     *      ROUTINES:
19     *      $RCHAR - READ A SINGLE CHARACTER
20     *      $WCHAR - WRITE A SINGLE CHARACTER
21     *      $PSEL - PRESET CONSOLE AND TAPE DEVS.
22     *      RIP - REASSIGN INPUT PORT
23     *      ROP - REASSIGN OUTPUT PORT
24     *      $PRP - PERMANENTLY REASSIGN PORTS
25     *
26     *
27     *      THE CONSOLE PACKAGE PROVIDES SOPHISTICATED SUPPORT FOR IT'S CALLERS:
28     *      SUPPORT FOR H8-4 AND/OR H8-5
29     *      INTERRUPT PROCESSING FOR INPUT CHARACTERS
30     *      28 CHARACTER TYPE-AHEAD CAPABILITY
31     *      40 CHARACTER TYPE-BEHIND CAPABILITY FOR H8-4 PORTS
32     *      SPECIAL CONTROL CHARACTER PROCESSING.
33     *
34     *
35     *
36     *
37     *
38     *
39     *      NOTE
40     *
41     *      IF THE CONSOLE PACKAGE IS USED BY ANY RUN-
42     *      NING ROUTINE, ALL ROUTINES RUNNING WITH IT
43     *      MUST USE IT ALSO. THIS IS BECAUSE
44     *      *CONSL1* WILL PROCESS INPUT CHARACTERS AT
45     *      INTERRUPT TIME, BEATING OUT ANY TASK-TIME
46     *      ROUTINE WHICH ATTEMPTS TO READ CHARACTERS.
47     *
48     *
49     *
50     *
51     *      NOTE
52     *
53     *      UNLESS OTHERWISE INDICATED IN THE SUBROUTINE
54     *      HEADERS, ALL ROUTINES MUST BE CALLED
55     *      WITH INTERRUPTS ENABLED, AND ALL ROUTINES
56     *      EXIT WITH INTERRUPTS ENABLED.
57     *
58     *

```

```

60 ** SPECIAL CHARACTER PROCESSING.
61 *
62 * *CONSL1* SUPPORTS 8 SPECIAL CHARACTERS:
63 *
64 * CTL-A USER DEFINED CONTROL FLAGS. THESE CAN BE CHECKED
65 * AT TASK TIME, OR THE USER PROGRAM CAN SETUP AN
66 * INTERRUPT VECTOR WHICH IS ENTERED AT INTERRUPT
67 * TIME, WHEN ANY OF THESE CHARACTERS ARE ENTERED.
68 *
69 * CTL-O TOGGLE DISCARDING OUTPUT CHARACTERS
70 * CTL-P RESUME OUTPUTTING CHARACTERS
71 * CTL-S SUSPEND OUTPUT
72 * CTL-Q RESUME OUTPUT
73 *
74 * NOTE:
75 * PORT 3400 SUPPORTS ONLY A HARDWARE HANDSHAKE
76 * BASED ON AN INVERTED CTS SIGNAL. ALL OTHER
77 * INPUT FROM PORT 3400 WILL BE IGNORED.

```

```

79 ** CONSOLE DRIVER ASSEMBLY CONSTANTS.
80 *
81 *
82 *

```

```

84 ** #CSCTL (CONSOLE CONTROL FLAG) BITS.
85 *
86 * THESE BITS ARE SET IN #CSCTL WHEN THE APPROPRIATE CONTROL
87 * CHARACTERS ARE STRUCK.
88 *
89 * THESE CAN BE EXAMINED AT TASK-TIME, OR WHEN THE USER'S CONTROL
90 * CHARACTER ROUTINE IS ENTERED (AT INTERRUPT TIME)
91 *
92 *
93 CC.HLD EQU 01 SUSPEND OUTPUT
94 CC.DMP EQU 02 DISCARD OUTPUT
95 CC.CTLA EQU 0100 CTL-A
96 CC.CTLB EQU 0200 CTL-B
97 CC.CTLC EQU 0400 CTL-C
98 CC.CTLD EQU 1000 CTL-D

```

CONSOLE DRIVER DOCUMENTATION - #XX.05.XX SOFTWARE HEATH HBASM V1.4 01/20/78 PAGE 3
 SYSTEM I/O DRIVER ENTRY 16:23:05 11-JAN-79

```

101 ** REMOTE ENTRY POINTS FOR CONSOLE DRIVER;
102 *
103
104
040.106 --- --- --- 105 $RCHAR JMP $RCHAR READ ONE CHARACTER
106
040.111 --- --- --- 107 $WCHAR JMP $WCHAR WRITE ONE CHARACTER
108
040.114 --- --- --- 109 $PRSCJ JMP $PRSCJ PRESET CONSOLE
110
040.346 --- --- --- 111 $RET EQU 040346A DUMMY ROUTINE THAT MERELY RETURNS
112
113 ** DATA AND BUFFERS;
114 *
115
116
040.117 000 117 $INBUF DB 0 INPUT BUFFER COUNT
040.120 118 DS 30
000.035 119 $INBUFL EQU *- $INBUF-2 MAX LENGTH OF BUFFER
120
040.156 121 DS 28 RESERVED
122
040.212 346 040 123 $CSIC DW $RET ADDRESS OF USER ROUTINE FOR CTL-A THROUGH CTL-D
124
040.214 000 125 $CSCTL DB 0 CONSOLE CONTROL BYTE
040.215 000 126 COLNO DB 0 CONSOLE COLUMN NUMBER
040.216 120 127 $CSLEN DB 80 CONSOLE LENGTH
128
040.217 000 000 129 $CSLSTP DW 0 STOP BIT FLAG IFF HB-5 (2000000A IFF 2 STOP BITS)
130
000.040 131 SET $INBUF/1000A
000.000 132 ERNZ */1000A- ALL DATA MUST BE IN SAME PAGE
133
040.221 134 COBFWA DS 40D HB-4 BUFFER
040.271 135 COBLWA EQU * CONSOLE OUTPUT BUFFER LWA
040.271 221 040 136 COB.IN DW COBFWA
040.273 221 040 137 COB.OUT DW COBFWA
138
000.001 139 IDI.50 EQU 00000001B PORT IS 00D IFF 8250

```



```

143 **      $RCHAR - READ SINGLE CHARACTER.
144 *
145 *      $RCHAR IS CALLED TO READ A CONSOLE CHARACTER.
146 *
147 *      ENTRY  NONE
148 *      EXIT   (A) = CHAR
149 *      INTERRUPTS ENABLED
150 *      USES   A,F
151
152
153 $RCHAR. EQU  *
```

```

155 **      $WCHAR - WRITE SINGLE CHARACTER.
156 *
157 *      $WCHAR IS CALLED TO OUTPUT A SINGLE CHARACTER.
158 *
159 *      ENTRY  (A) = CHARACTER
160 *      EXIT   INTERRUPTS ENABLED
161 *      USES   NONE
162
163
164 $WCHAR. EQU  *
```




```

204 *** $PRSC - PRESET CONSOLE.
205 *
206 * $PRSC TURNS OFF ALL I/O PORTS, AND SETS UP INPUT BACK TO
207 * THE MASTER PORT.
208 *
209 * ENTRY (A) = NEW PORT NUMBER (+IDE.50 IFF 8250)
210 * (HL)[0-14] = NEW BAUD RATE
211 * (HL)[15] = 1 IF TWO STOP BITS
212 * EXIT NONE
213
214
215 $PRSC EQU *
---

```



CONSOLE DRIVER DOCUMENTATION - #XX.05.XX SOFTWARE HEATH HRASH V1.4 01/20/78 PAGE 7
 *PRSC - PRESET CONSOLE 16:23:06 11-JAN-79

```

242          LON      C
243 *****
244 *****
245 **
246 **      THIS IS THE END OF THE CONSOLE DRIVER PACKAGE. THE FOLLOWING
247 **      ROUTINES ARE NOT PART OF THE CONSOLE DRIVER PACKAGE, BUT ARE
248 **      SUPPLIED FOR LOOKING UP THE DESIRED BAUD RATE RATE VALUES
249 **      AS REQUIRED BY THE CONSOLE PACKAGE ROUTINES. THE USER MUST
250 **      CODE THEM HIMSELF IF HE WANTS TO USE THEM.
251 **
252 *****
253 *****
254
255
044.125      256          XTEXT      LRD

258X ***      $LRD - LOOKUP BAUDRATE DIVISOR.
259X *
260X *      $LRD TRANSLATES A BAUD RATE INTO THE PROPER DIVISOR FOR THE
261X *      8250 CHIPS ON THE HR-4 SERIAL CARD.
262X *
263X *      NOTE THAT $LRD DOES NOT ACTUALLY COMPUTE THE TRANSFORMATION, BUT
264X *      SIMPLY LOOKS UP THE VALUE IN A TABLE. THIS IS DONE TO DETECT TYPOS
265X *      IN THE USER SUPPLIED BAUD RATE.
266X *
267X *      ENTRY (DE) = BAUD RATE (AS A BINARY NUMBER)
268X *      EXIT (Z) SET IF VALID BAUD RATE
269X *      (HL) = DIVISOR
270X *      (Z) CLEAR IF NOT VALID BAUD RATE
271X *      USES A,F,D,E,H,L
272X
273X
044.125 172      274X $LRD      MOV      A,D
044.126 263      275X          ORA      E          (A) = CODE VALUE
044.127 041 142 044 276X          LXI      H,LBDA      (HL) = LOOKUP TABLE
044.132 315 207 044 277X          CALL   $WTBLS      WORD TABLE LOOKUP
044.135 176      278X          MOV      A,M
044.136 043      279X          INC      H
044.137 146      280X          MOV      H,M
044.140 157      281X          MOV      L,A
044.141 311      282X          RET          RETURN WITH CONDITION CODE FROM $WTBLS
283X
284X
285X **      BAUD RATE VS 8250 DIVISOR TABLE.
286X *
287X *      KEY IS BAUD RATE SQUEEZED INTO ONE BYTE
288X
044.142      289X LBDA      DS      0
044.142 151      290X          DB      2400/256!#2400
044.143 060 000      291X          DW      000060A
044.145 245      292X          DB      9600/256!#9600
044.146 014 000      293X          DW      000014A
044.150 132      294X          DB      600/256!#600

```

*PRSCCL - PRESET CONSOLE

*LRD

10:23:07...11-JAN-79

```

044.151 300 000 295X DW 000300A
044.153 113 000 296X DB 19200/256!*19200
044.154 006 000 297X DW 000006A
044.156 322 000 298X DB 4800/256!*4800
044.157 030 000 299X DW 000030A
044.161 264 000 300X DB 1200/256!*1200
044.162 140 000 301X DW 000140A
044.164 055 000 302X DB 300/256!*300
044.165 200 001 303X DW 001200A
044.167 074 000 304X DB 7200/256!*7200
044.170 020 000 305X DW 000020A
044.172 036 000 306X DB 3600/256!*3600
044.173 040 000 307X DW 000040A
044.175 017 000 308X DB 1800/256!*1800
044.176 100 000 309X DW 000100A
044.200 156 000 310X DB 110/256!*110
044.201 027 204 311X DW 204027A
044.203 226 000 312X DB 150/256!*150
044.204 000 003 313X DW 003000A
044.206 000 000 314X DB 0
                                END OF TABLE
044.207 315
                                316 XTEXT WTBLS

318X ** $WTBLS - TABLE SEARCH
319X *
320X * $WTBLS LOOKS UP WORD VALUES IN A TABLE, USING A ONE-BYTE
321X * KEY.
322X *
323X * TABLE FORMAT
324X *
325X * DB KEY1
326X * DW VAL1
327X *
328X *
329X * DB KEYN
330X * DW VALN
331X * DB 0
332X *
333X * ENTRY (A) = PATTERN
334X * (H,L) = TABLE FWA
335X * EXIT (A) = PATTERN IF FOUND
336X * 'Z' SET IF FOUND
337X * USES A,F,H,L
338X
339X
044.207 305 340X $WTBLS PUSH B
044.210 107 341X MOV B,A
044.211 176 342X $WTBL1 MOV A,M (A) = CHARACTER
044.212 043 343X INX H
044.213 270 344X CMP B
044.214 312 231 044 345X JZ $WTBL2 IF MATC
044.217 247 346X ANA A
044.220 043 347X INX H

```

CONSOLE DRIVER DOCUMENTATION - *XX.05.XX SOFTWARE

HEATH HBASM V1.4 01/20/78

PAGE 9

*PRSC - PRESET CONSOLE

*WTBLS

16:23:07 11-JAN-79

```
044.221 043      348X      INX      H      SKIP PAST
044.222 302 211 044 349X      JNZ      $WTBL1  IF NOT END OF TABLE
044.225 053      350X      DCX      H
044.226 053      351X      DCX      H
044.227 053      352X      DCX      H
044.230 264      353X      ORA      H      CLEAR 'Z'
          354X
          355X *      DONE
          356X
044.231 301      357X $WTBL2 POP      B
044.232 311      358X      RET
          359
044.233      360      END
ASSEMBLY COMPLETE
360 STATEMENTS
0 ERRORS DETECTED
15568 BYTES FREE
```

CROSS REFERENCE TABLE

PAGE 10

\$CSIC	040212	123L		
\$CSLCTL	040214	125L		
\$CSLLEN	040216	127L		
\$CSLSTP	040217	129L		
\$INBUF	040117	117L	119	131
\$INBUFL	000035	119E		
\$LBD	044125	274L		
\$PRF	043035	228E		
\$PRSCL	040114	109L		
\$PRSCL.	-----	109	215E	
\$RCHAR	040106	105L		
\$RCHAR.	-----	105	153E	
\$RET	040346	111E	123	
\$WCHAR	040111	107L		
\$WCHAR.	-----	107	164E	
\$WTBL1	044211	342L	349	
\$WTBL2	044231	345	357L	
\$WTBLS	044207	277	340L	
.	000040	131S	132	
CC.CTLA	000010	95E		
CC.CTLB	000020	96E		
CC.CTLC	000040	97E		
CC.CTLD	000100	98E		
CC.DMP	000002	94E		
CC.HLD	000001	93E		
COB.IN	040271	136L		
COB.OUT	040273	137L		
COBFWA	040221	134L	136	137
COBLWA	040271	135E		
COLNO	040215	126L		
ENDCNSL	044125	239E		
IDF.50	000001	139E		
LEDA	044142	276	289L	
RIP	042046	183E		
ROP	042177	200E		

31778 BYTES FREE

APPENDIX C

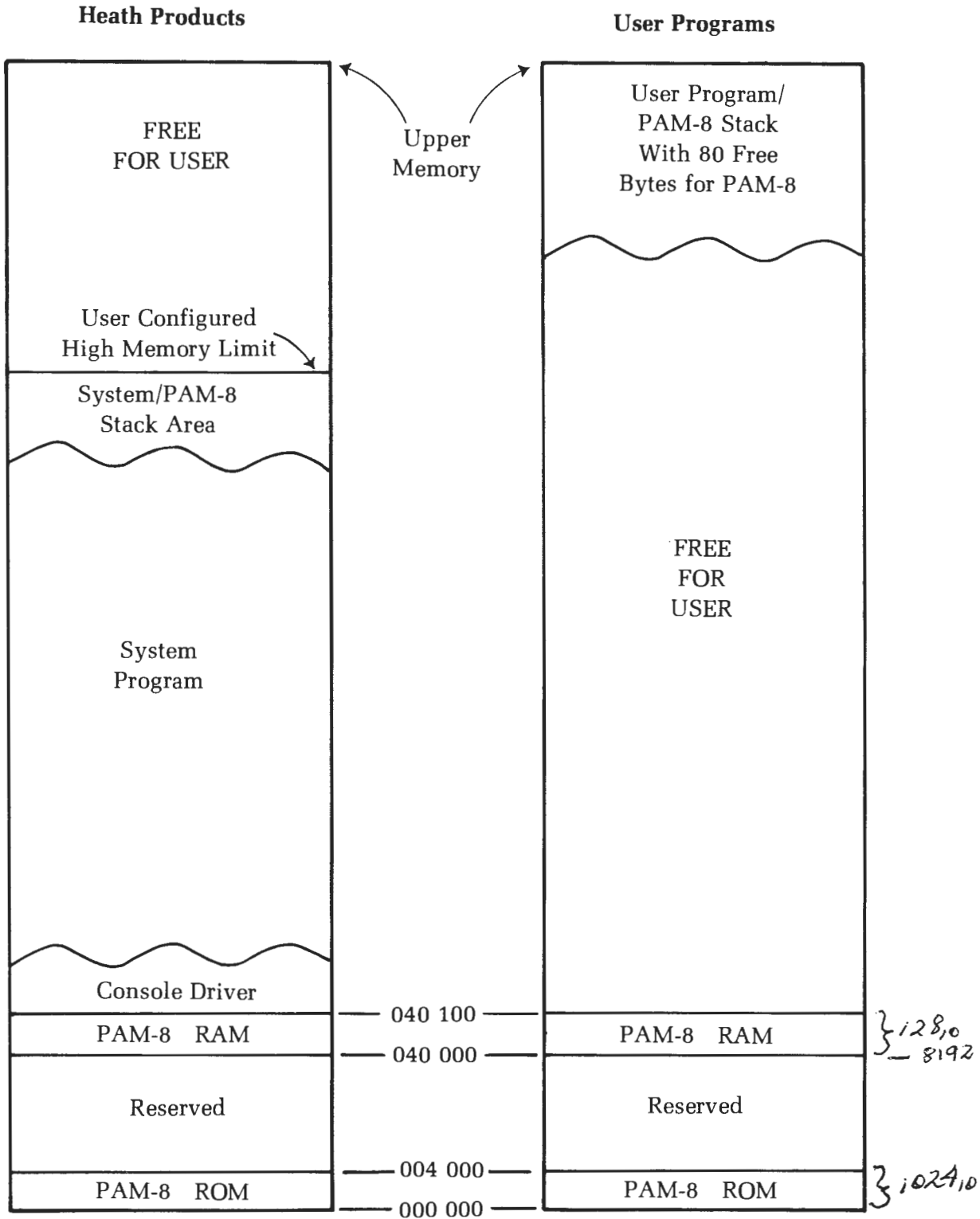
This appendix contains I/O and Memory Maps for the H8 software products.

I/O map

377	Reserved
376	Reserved
375	Aux. Terminal
374	Aux. Terminal
373	Console Control
372	Console Data
371	Load/Dump Control
370	Load/Dump Data
≈ Reserved ≈	
361	F.P Segments
360	F.P Digit & Key
357 thru 350	H8-4 Console Driver
347 thru 340	H8-4 Line Printer
≈ Reserved ≈	
307 thru 300	H8-4 Auxiliary Terminal
≈ Reserved ≈	
177 thru 174	H17
173 thru 100	Reserved
077 thru 000	User Available I/O Ports

NOTE: The H8-5 Serial I/O and Cassette Interface uses port locations 370 thru 375.

MEMORY MAP



APPENDIX D

ASCII Characters

<u>7-BIT OCTAL CODE</u>	<u>DECIMAL CODE</u>	<u>CHARACTER</u>	<u>DESCRIPTION</u>
000	0	NUL	NULL, TAPE FEED, CONTROL SHIFT P.
001	1	SOH	START OF HEADING; ALSO SOM, START OF MESSAGE, CONTROL A,
002	2	STX	START OF TEXT; ALSO EOA, END OF ADDRESS, CONTROL B,
003	3	ETX	END OF TEXT; ALSO EOM, END OF MESSAGE CONTROL C,
004	4	EOT	END OF TRANSMISSION (END); CONTROL D,
005	5	ENQ	ENQUIRY; ALSO WRU, CONTROL E,
006	6	ACK	ACKNOWLEDGE. ALSO RU, CONTROL F.
007	7	BEL	RINGS THE BELL. CONTROL G.
010	8	BS	BACKSPACE; ALSO FEO, FORMAT EFFECTOR BACKSPACE SOME MACHINES, CONTROL H.
011	9	HT	HORIZONTAL TAB. CONTROL I
012	10	LF	LINE FEED (NEW LINE); ADVANCES PAPER TO NEXT LINE, DUPLICATED BY CONTROL J.
013	11	VT	VERTICAL TAB (VTAB). CONTROL K.
014	12	FF	FORM FEED TO TOP OF NEXT PAGE (PAGE). CONTROL L.
015	13	CR	CARRIAGE RETURN TO BEGINNING OF LINE. DUPLICATED BY CONTROL M.
016	14	SO	SHIFT OUT: CHANGES RIBBON COLOR TO RED. CONTROL N.
017	15	SI	SHIFT IN: CHANGES RIBBON COLOR TO BLACK. CONTROL O.
020	16	DLE	DATA LINK ESCAPE. CONTROL P (DCO).
021	17	DC1	DEVICE CONTROL 1, TURNS TRANSMITTER (READER) ON, CONTROL Q (XON).
022	18	DC2	DEVICE CONTROL 2, TURNS PUNCH OR AUXILIARY ON, CONTROL R (TAPE, AUX ON).
023	19	DC3	DEVICE CONTROL 3, TURNS TRANSMITTER (READER) OFF, CONTROL S (XOFF).
024	20	DC4	DEVICE CONTROL 4, TURNS PUNCH OR AUXILIARY OFF. CONTROL T (TAPE, AUX OFF).
025	21	NAK	NEGATIVE ACKNOWLEDGE; ALSO ERR. ERROR. CONTROL U.
026	22	SYN	SYNCHRONOUS IDLE (SYNC). CONTROL V.
027	23	ETB	END OF TRANSMISSION BLOCK; ALSO LEM. LOGICAL END OF MEDIUM. CONTROL W.

7-BIT

OCTAL CODE	DECIMAL CODE	CHARACTER	DESCRIPTION
030	24	CAN	CANCEL (CANCL). CONTROL X.
031	25	EM	END OF MEDIUM. CONTROL Y.
032	26	SUB	SUBSTITUTE. CONTROL Z.
033	27	ESC	ESCAPE. PREFIX.
034	28	FS	FILE SEPARATOR. CONTROL SHIFT L.
035	29	GS	GROUP SEPARATOR. CONTROL SHIFT M.
036	30	RS	RECORD SEPARATOR. CONTROL SHIFT N.
037	31	US	UNIT SEPARATOR. CONTROL SHIFT O.
040	32	SP	SPACE.
041	33	!	
042	34	"	
043	35	#	
044	36	\$	
045	37	%	
046	38	&	
047	39	'	ACUTE ACCENT OR APOSTROPHE.
050	40	(
051	41)	
052	42	*	
053	43	+	
054	44	,	
055	45	-	
056	46	.	
057	47	/	
060	48	0	
061	49	1	
062	50	2	
063	51	3	
064	52	4	
065	53	5	
066	54	6	
067	55	7	
070	56	8	
071	57	9	
072	58	:	
073	59	;	
074	60	<	
075	61	=	
076	62	>	
077	63	?	
100	64	@	
101	65	A	
102	66	B	
103	67	C	
104	68	D	
105	69	E	
106	70	F	
107	71	G	

<u>7-BIT OCTAL CODE</u>	<u>DECIMAL CODE</u>	<u>CHARACTER</u>	<u>DESCRIPTION</u>
110	72	H	
111	73	I	
112	74	J	
113	75	K	
114	76	L	
115	77	M	
116	78	N	
117	79	O	
120	80	P	
121	81	Q	
122	82	R	
123	83	S	
124	84	T	
125	85	U	
126	86	V	
127	87	W	
130	88	X	
131	89	Y	
132	90	Z	
133	91	[SHIFT K
134	92]	SHIFT L
135	93	↑	SHIFT M
136	94	←	SHIFT N
137	95		
140	96		ACCENT GRAVE.
141	97	a	
142	98	b	
143	99	c	
144	100	d	
145	101	e	
146	102	f	
147	103	g	
150	104	h	
151	105	i	
152	106	j	
153	107	k	
154	108	l	
155	109	m	
156	110	n	
157	111	o	
160	112	p	
161	113	q	
162	114	r	
163	115	s	
164	116	t	
165	117	u	
166	118	v	
167	119	w	

7-BIT			
OCTAL	DECIMAL		
<u>CODE</u>	<u>CODE</u>	<u>CHARACTER</u>	<u>DESCRIPTION</u>
170	120	x	
171	121	y	
172	122	z	
173	123		
174	124		
175	125		THIS CODE GENERATED BY ALT MODE.
176	126		THIS CODE GENERATED BY ESC KEY (IF PRESENT)
177	127	DEL	DELETE, RUB OUT.

APPENDIX E

Number Conversion For 0 To 255₁₀

<u>DEC</u>	<u>HEX</u>	<u>OCT</u>	<u>BINARY</u>	<u>DEC</u>	<u>HEX</u>	<u>OCT</u>	<u>BINARY</u>
000	00	000	00000000	040	28	050	00101000
001	01	001	00000001	041	29	051	00101001
002	02	002	00000010	042	2A	052	00101010
003	03	003	00000011	043	2B	053	00101011
004	04	004	00000100	044	2C	054	00101100
005	05	005	00000101	045	2D	055	00101101
006	06	006	00000110	046	2E	056	00101110
007	07	007	00000111	047	2F	057	00101111
008	08	010	00001000	048	30	060	00110000
009	09	011	00001001	049	31	061	00110001
010	0A	012	00001010	050	32	062	00110010
011	0B	013	00001011	051	33	063	00110011
012	0C	014	00001100	052	34	064	00110100
013	0D	015	00001101	053	35	065	00110101
014	0E	016	00001110	054	36	066	00110110
015	0F	017	00001111	055	37	067	00110111
016	10	020	00010000	056	38	070	00111000
017	11	021	00010001	057	39	071	00111001
018	12	022	00010010	058	3A	072	00111010
019	13	023	00010011	059	3B	073	00111011
020	14	024	00010100	060	3C	074	00111100
021	15	025	00010101	061	3D	075	00111101
022	16	026	00010110	062	3E	076	00111110
023	17	027	00010111	063	3F	077	00111111
024	18	030	00011000	064	40	100	01000000
025	19	031	00011001	065	41	101	01000001
026	1A	032	00011010	066	42	102	01000010
027	1B	033	00011011	067	43	103	01000011
028	1C	034	00011100	068	44	104	01000100
029	1D	035	00011101	069	45	105	01000101
030	1E	036	00011110	070	46	106	01000110
031	1F	037	00011111	071	47	107	01000111
032	20	040	00100000	072	48	110	01001000
033	21	041	00100001	073	49	111	01001001
034	22	042	00100010	074	4A	112	01001010
035	23	043	00100011	075	4B	113	01001011
036	24	044	00100100	076	4C	114	01001100
037	25	045	00100101	077	4D	115	01001101
038	26	046	00100110	078	4E	116	01001110
039	27	047	00100111	079	4F	117	01001111

DEC	HEX	OCT	BINARY
080	50	120	01010000
081	51	121	01010001
082	52	122	01010010
083	53	123	01010011
084	54	124	01010100
085	55	125	01010101
086	56	126	01010110
087	57	127	01010111
088	58	130	01011000
089	59	131	01011001
090	5A	132	01011010
091	5B	133	01011011
092	5C	134	01011100
093	5D	135	01011101
094	5E	136	01011110
095	5F	137	01011111
096	60	140	01100000
097	61	141	01100001
098	62	142	01100010
099	63	143	01100011
100	64	144	01100100
101	65	145	01100101
102	66	146	01100110
103	67	147	01100111
104	68	150	01101000
105	69	151	01101001
106	6A	152	01101010
107	6B	153	01101011
108	6C	154	01101100
109	6D	155	01101101
110	6E	156	01101110
111	6F	157	01101111
112	70	160	01110000
113	71	161	01110001
114	72	162	01110010
115	73	163	01110011
116	74	164	01110100
117	75	165	01110101
118	76	166	01110110
119	77	167	01110111
120	78	170	01111000
121	79	171	01111001
122	7A	172	01111010
123	7B	173	01111011

DEC	HEX	OCT	BINARY
124	7C	174	01111100
125	7D	175	01111101
126	7E	176	01111110
127	7F	177	01111111
128	80	200	10000000
129	81	201	10000001
130	82	202	10000010
131	83	203	10000011
132	84	204	10000100
133	85	205	10000101
134	86	206	10000110
135	87	207	10000111
136	88	210	10001000
137	89	211	10001001
138	8A	212	10001010
139	8B	213	10001011
140	8C	214	10001100
141	8D	215	10001101
142	8E	216	10001110
143	8F	217	10001111
144	90	220	10010000
145	91	221	10010001
146	92	222	10010010
147	93	223	10010011
148	94	224	10010100
149	95	225	10010101
150	96	226	10010110
151	97	227	10010111
152	98	230	10011000
153	99	231	10011001
154	9A	232	10011010
155	9B	233	10011011
156	9C	234	10011100
157	9D	235	10011101
158	9E	236	10011110
159	9F	237	10011111
160	A0	240	10100000
161	A1	241	10100001
162	A2	242	10100010
163	A3	243	10100011
164	A4	244	10100100
165	A5	245	10100101
166	A6	246	10100110
167	A7	247	10100111

<u>DEC</u>	<u>HEX</u>	<u>OCT</u>	<u>BINARY</u>	<u>DEC</u>	<u>HEX</u>	<u>OCT</u>	<u>BINARY</u>
168	A8	250	10101000	212	D4	324	11010100
169	A9	251	10101001	213	D5	325	11010101
170	AA	252	10101010	214	D6	326	11010110
171	AB	253	10101011	215	D7	327	11010111
172	AC	254	10101100	216	D8	330	11011000
173	AD	255	10101101	217	D9	331	11011001
174	AE	256	10101110	218	DA	332	11011010
175	AF	257	10101111	219	DB	333	11011011
176	B0	260	10110000	220	DC	334	11011100
177	B1	261	10110001	221	DD	335	11011101
178	B2	262	10110010	222	DE	336	11011110
179	B3	263	10110011	223	DF	337	11011111
180	B4	264	10110100	224	E0	340	11100000
181	B5	265	10110101	225	E1	341	11100001
182	B6	266	10110110	226	E2	342	11100010
183	B7	267	10110111	227	E3	343	11100011
184	B8	270	10111000	228	E4	344	11100100
185	B9	271	10111001	229	E5	345	11100101
186	BA	272	10111010	230	E6	346	11100110
187	BB	273	10111011	231	E7	347	11100111
188	BC	274	10111100	232	E8	350	11101000
189	BD	275	10111101	233	E9	351	11101001
190	BE	276	10111110	234	EA	352	11101010
191	BF	277	10111111	235	EB	353	11101011
192	C0	300	11000000	236	EC	354	11101100
193	C1	301	11000001	237	ED	355	11101101
194	C2	302	11000010	238	EE	356	11101110
195	C3	303	11000011	239	EF	357	11101111
196	C4	304	11000100	240	F0	360	11110000
197	C5	305	11000101	241	F1	361	11110001
198	C6	306	11000110	242	F2	362	11110010
199	C7	307	11000111	243	F3	363	11110011
200	C8	310	11001000	244	F4	364	11110100
201	C9	311	11001001	245	F5	365	11110101
202	CA	312	11001010	246	F6	366	11110110
203	CB	313	11001011	247	F7	367	11110111
204	CC	314	11001100	248	F8	370	11111000
205	CD	315	11001101	249	F9	371	11111001
206	CE	316	11001110	250	FA	372	11111010
207	CF	317	11001111	251	FB	373	11111011
208	D0	320	11010000	252	FC	374	11111100
209	D1	321	11010001	253	FD	375	11111101
210	D2	322	11010010	254	FE	376	11111110
211	D3	323	11010011	255	FF	377	11111111

APPENDIX F

Memory Table

Offset Octal and Decimal Boundaries

<u>Hi Byte</u>	<u>Lo Byte</u>	<u>Decimal Boundary</u>
A15.....A8	A7.....A0	
0 0 4	0 0 0	1024
0 2 0	0 0 0	4096
0 4 0	0 0 0	8192
0 6 0	0 0 0	12288
1 0 0	0 0 0	16384
1 2 0	0 0 0	20480
1 4 0	0 0 0	24576
1 6 0	0 0 0	28672
2 0 0	0 0 0	32768
2 2 0	0 0 0	36864
2 4 0	0 0 0	40960
2 6 0	0 0 0	45056
3 0 0	0 0 0	49152
3 2 0	0 0 0	53248
3 4 0	0 0 0	57344
3 6 0	0 0 0	61440
3 7 7	3 7 7	65535*

For example, if you have 12K bytes in an H8, the lower boundary is at 8192, or 040 000 offset octal. The upper boundary is at $8K + 12K = 20K$ (20480), or 120 000 Octal.

*NOTE: 65,535 is the last location in a memory addressed by 16 bits.

INDEX

- ASCII Characters, 0-50
- ASR Console, 0-30
- Basic Program Format, 0-19
- Checksum, 0-18
- Compressed, Text Format, 0-19
- Console Debugger (BUG-8), 0-11
- Console Driver, 0-30, 0-33
- Console Driver Documentation, 0-37
- Copying Tapes, 0-29
- CRC-16, 0-17
- Control Characters, 0-32
- Data Formats, 0-19
- Displays, 0-20
- Entry Point, 0-33
- File, 0-17
- Front Panel Displays, 0-20
- Front Panel Entry Points, 0-33
- Heath Assembly Language (HASL-8), 0-14
- Heath Text Editor (TED-8), 0-12
- H8-4 Port Allocation, 0-31
- Installation, 0-25 ff.,
- Installing a Patch, 0-35
- Label Record, 0-17
- Loading, 0-25
- Magnetic Tape, 0-22
- Memory Image Format, 0-19
- Number Conversions, 0-54
- Options, 0-26
- Patch Installation, 0-35
- Pad Characters, 0-26
- Panel Monitor (PAM-8), 0-10
- Product Installation, 0-25
- ROM, 0-10
- Record, 0-17
- Record Structure, 0-17
- Software Installation, 0-25
- Software Distribution Tape, 0-25
- Software Problems, 0-34
- Software Version Code, 0-32
- System Configuration, 0-7
- Tape Files, 0-17
- Tape Format, 0-17
- 110-Baud Console, 0-30

Section 1

PANEL MONITOR

PAM-8



TABLE OF CONTENTS

INTRODUCTION	1-4
THEORY OF OPERATION	
Power Up and Master Clear	1-5
Clock Interrupts	1-5
PAM-8 Modes/Using RST and RTM	1-6
H8 Displays	1-7
H8 Keypad	1-9
DISPLAYING AND ALTERING MEMORY LOCATIONS	
Specifying a Memory Address	1-10
Altering a Memory Location	1-12
Stepping Through Memory	1-13
DISPLAYING AND ALTERING REGISTERS	
Specifying a Register for Display	1-14
Altering the Contents of a Selected Register	1-15
Stepping Through the Registers	1-15
PROGRAM EXECUTION CONTROL	
Initiating Program Execution	1-16
Breakpointing	1-16
Single Instruction Operation	1-17
Interrupting a Program During Execution	1-17

LOAD/DUMP ROUTINES	1-18
Loading From Tape	1-18
Dumping to Tape	1-18
Copying a Tape	1-20
Tape Errors	1-20
 I/O FACILITIES	
Inputting From a Port	1-21
Outputting to a Port	1-21
Addressing Port Pairs	1-21
 ADVANCED CONTROL	
16-Bit Tick Counter (TICCNT)	1-22
Using the Keypad	1-22
Display Usage	1-23
Using Interrupts	1-24
 APPENDIX A	1-25
 APPENDIX B	1-64
 INDEX	1-68



INTRODUCTION

This Manual describes the functions and operations of the Heath H8 Panel Monitor Program, PAM-8, which resides permanently in a ROM on the H8 CPU board. PAM-8 provides a sophisticated front panel display and keyboard emulation as well as handling master clear and interrupt operations. Some of the major features of PAM-8 are:

- Memory contents display and alteration.
- Register contents display and alteration.
- Program execution control (both breakpoint and single instruction operation).
- Self-contained bootstraps for program loading and dumping.
- Port input and output routines.

In addition to the above features, PAM-8 can be instructed (by means of a flag byte contained in H8 RAM) to bypass some or all of its normal functions so the sophisticated user can augment or totally replace them.

Communication with the Panel Monitor is accomplished through three devices: the keypad, the 7-segment displays, and the audio alert. The user enters commands and values through the 16-key keypad, and PAM-8 responds visually through the front panel displays. In addition to the front panel displays, PAM-8 provides the keypad entry and function feedback to the built-in speaker. Appropriate signals (short, medium, and long beeps) indicate that commands and data are accepted or rejected.

THEORY OF OPERATION

This section will supplement the information contained in the "Operation" and "Circuit Description" sections of your H8 Operation Manual. In order to fully understand how PAM-8 operates, you must be familiar with the H8 front panel and CPU. A thorough knowledge of the 8080 instruction set and its architecture is also essential.

Power Up and Master Clear

PAM-8 initializes the H8 whenever you power-up or master clear (RST). You initiate the power-up operation by turning on the rear panel Power switch. You can master clear by simultaneously depressing both the lower right-hand (RST/Ø) and lower left-hand (Ø) keys of the H8 front panel keypad. Both power-up and RST cause a level zero (highest priority) interrupt and result in a long beep from the audio alert.

During initialization, PAM-8 enters a routine which determines the high limit of continuous RAM. Once the high limit of available RAM is determined, the H8 stack pointer (SP) is set to this value and control is passed to the front panel command loop. Using this feature, you can immediately determine the total amount of continuous memory above 8K by displaying stack pointer value.

Clock Interrupts

The Clock Interrupt is a crucial element in the operation of the H8 front panel system. This level one interrupt is generated by the front panel hardware every 2,000 μ S. PAM-8 uses this interrupt to check for some keyboard commands, to check for user program breakpoints, and to refresh the front panel displays.

PAM-8 performs these functions using a series of subroutines which are executed as necessary when indicated by the interrupts. For this reason, all user programs must maintain a valid stack (at high memory) containing at least 80 free bytes at all times. If this stack space is not available and PAM-8 is running (it can be disabled; see the Advanced Control Section), unpredictable software damage can occur in your program. In the same manner, if your program should execute a DI (Disable Interrupt) instruction, no front panel services including the RTM (Return To Monitor) function are available until an EI (Enable Interrupt) instruction is executed or until a master clear (RST/Ø) is performed.



PAM-8 Modes/Using RST and RTM

PAM-8 is always in either the monitor mode or the user mode. In the monitor mode no user program is executing, PAM-8 loops reading the keypad and refreshing the displays. All commands entered via the keypad are valid; however, the RTM command is meaningless.

When your program is being executed, PAM-8 is in the user mode and the MON LED on the front panel is extinguished. Only two keyboard commands are valid in this mode: RST (master clear) and RTM (Return To Monitor). NOTE: Both of these commands are dual key commands. No single key command is recognized, so a user program may have free use of the entire keypad.

You can return PAM-8 to the monitor mode by using the RTM command (simultaneously press the \emptyset and the # keys). This command stops program execution at the end of the current instruction, stores the current value of each register, and returns PAM-8 to the monitor mode. You can then continue your program by pressing the GO key. The RST command (simultaneously press the 0 and the / keys) performs the master clear operation described earlier and does not save any register values.

Normally, when a user program is running, PAM-8 is also running. Thus, if PAM-8 is displaying the contents of the HL register pair and the user program is started, it continues to display the contents of this register pair as the program is run. If the user program changes the contents of the HL pair, the change is immediately reflected in the front panel displays. In a similar manner, if a memory location is displayed when a user program is started, it is displayed during the time the user program is run. If the user program changes the contents of the displayed memory location, the front panel display changes.

Since PAM-8 does not recognize keypad commands in the user mode, the RTM command must be used before the memory location or register being displayed is changed to a new location or a different register. Once you select the new location or different register, you can resume program execution by pressing GO.

NOTE: PAM-8 requires about 10% of the H8 CPU's resources to process the display interrupts. Programs which are compute-bound may be slowed down by simultaneous operation of PAM-8. In this situation, you may wish to turn off the clock interrupts to improve execution time. See "Using Interrupts" on Page 1-24.

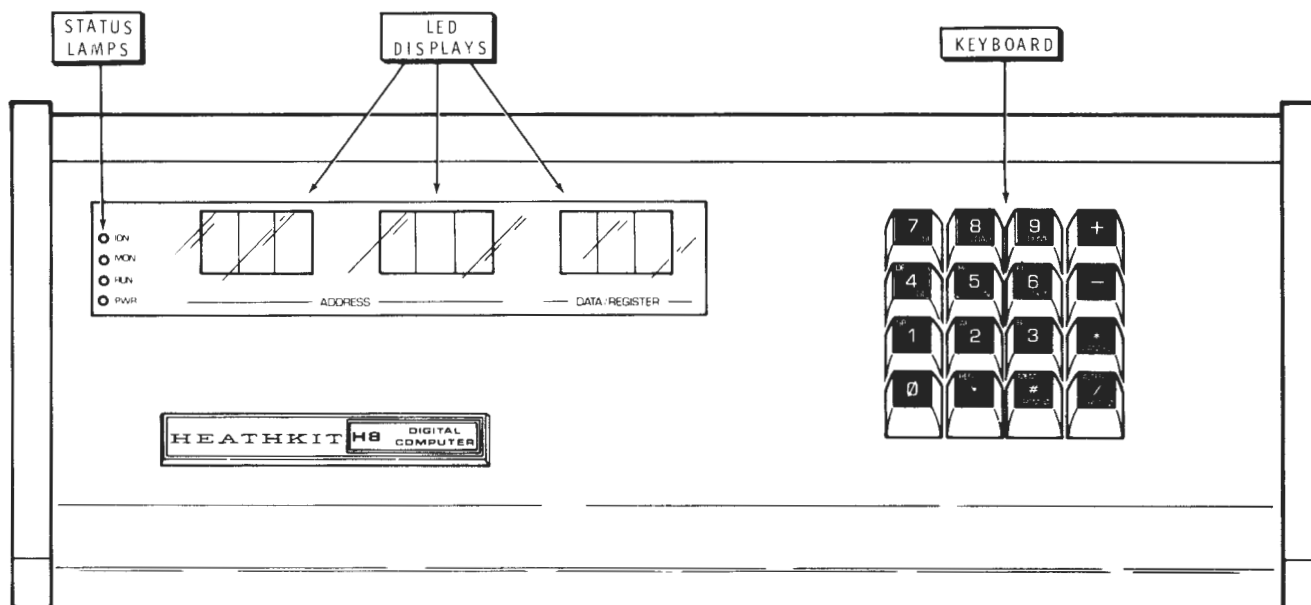


Figure 1-1

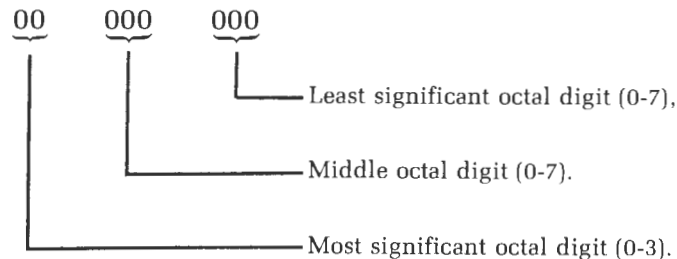
H8 Displays

You must understand the H8 front panel presentation in order to use PAM-8. The display is made up of 9 digits, in three groups of three digits each. See Figure 1-1. Each group of three digits displays one byte (eight bits) of information. This information may be the contents of a designated register or memory location, or it may be the address of a memory location itself. The register names are also displayed.

All binary numbers are converted to octal format for display on the H8 front panel. The following table shows binary to octal conversion.

<u>BINARY NUMBER</u>	<u>OCTAL NUMBER</u>
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Each byte is displayed as two-and-one-half octal digits. The octal numbers lie in the range of 000 to 377 for binary numbers in the range 00000000 to 11111111, as shown below.



NOTE: As there are only eight bits in a byte, the most significant octal digit only represents two bits and is therefore displayed as 0 to 3. If the user should inadvertently enter the octal digits 4 to 7 into the most significant digit, the most significant bit is lost. Losing this bit converts 4 through 7 into the digits 0 through 3 respectively.

Also note that 16-bit numbers, such as memory addresses and certain register contents, are still displayed as two eight-bit numbers. Therefore, the H8 front panel representation of the number is made up of **two** groups of three octal numbers in the range of 000 to 377. This representation of 16-bit binary numbers is known as **offset octal**, and is used consistently throughout all H8 displays of 16-bit numbers. Offset octal must not be confused with octal. For example:

$\begin{array}{cccccc} \overline{111111} & \overline{111111} & & \overline{111111} & \overline{111111} \\ & & & & & \\ 3 & 7 & 7 & 3 & 7 & 7 \end{array}$	<p>A 16-bit binary number</p> <p>Offset octal representation (377 377)</p>
---	--

$\begin{array}{cccccc} \overline{111111} & \overline{111111} & \overline{111111} & \overline{111111} & \overline{111111} & \overline{111111} \\ & & & & & \\ 1 & 7 & 7 & 7 & 7 & 7 \end{array}$	<p>A 16-bit binary number</p> <p>True Octal representation (177777)</p>
---	---

The lower example shows true octal representation of a 16-bit binary number. This is **not** used by the H8 front panel displays or any H8 software. Occasionally you will see offset octal numbers printed with a decimal point separating the upper and lower bytes. For example:

$\overline{377.377}$	
Hi Byte	Lo Byte

H8 Keypad

The H8 Keypad consists of 16 keys, as shown in Figure 1-1. When the keypad is operating under the control of PAM-8, it exhibits a number of unique properties.

- Each keystroke is verified by a short beep from the audio alert.
- Octal digits are entered using the keys 0 through 7.
- Holding a key down continuously repeats the key's function.
- The + key increments memory port or register locations.
- The - key decrements memory port or register locations.
- The * key cancels previous keypad entries.
- The ALTER key causes PAM-8 to enter the alter mode.
- The MEM key causes PAM-8 to enter the display memory mode.
- The REG key causes PAM-8 to enter the register mode.

Many of the keys on the keypad have multiple functions, depending on the PAM-8 mode being used. In the register mode, for example, the numeric keys (1-6) call the register indicated in the upper left-hand corner of the key. When the PAM-8 is in neither the register nor the memory mode, the keys perform the functions indicated in the lower right-hand corner of the key.

The # and / keys have additional special functions, as indicated earlier. When the / key is pressed simultaneously with the 0 key, the RST (master clear) sequence is initiated. When the # sign key is depressed simultaneously with the 0 key, the RTM (Return To Monitor) function is initiated, the user program is stopped, and PAM-8 regains control.

Each key is covered in greater detail as the various function are discussed.



DISPLAYING AND ALTERING MEMORY LOCATIONS

One of the major features of PAM-8 is its ability to examine the contents of any H8 memory location and to modify the contents of that memory location if it is RAM.

When the H8 is first powered up, PAM-8 is in the display memory mode. This mode is indicated by all digits displaying octal numbers and no decimal points being on.

Specifying a Memory Address

If you wish to display or alter the contents of a memory location. You must first place PAM-8 in the memory address mode and then enter the desired memory address. Place PAM-8 in the memory address mode (if not already there) by pressing the MEM (Memory) key. Specify the address to be displayed or altered by entering the 6-digit address (offset octal).

When you press the MEM key, all the decimal points will light. This indicates that the address may now be entered. Once the full 6-digit address is entered, the decimal points turn off, indicating that address entry is completed. After all 6 digits are entered, the address is displayed in the left-most six displays, and the contents of the addressed memory location are displayed in the right-hand 3 digits.

NOTE: As you press each key, including the MEM key, a short beep indicates successful entry. As each group of three octal digits is successfully entered, a medium beep is sounded. The sequence by which you specify a memory address is shown in Figure 1-2.

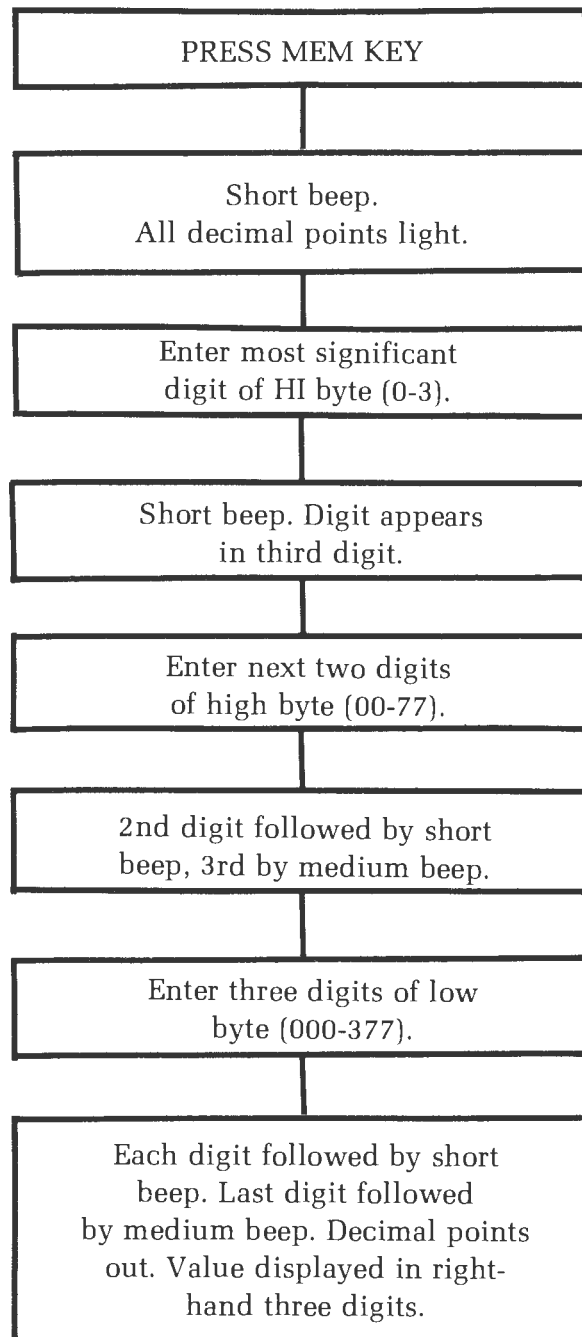


Figure 1-2

Entering a memory address through PAM-8.

NOTE: If you press a non-octal digit key as one of the six address digits, an error is flagged (a long beep). Once this error is flagged, the PAM-8 considers the address complete and extinguishes the decimal points. The entire sequence must be repeated.

Altering a Memory Location

Before you can alter a memory location, you must first display the contents of the memory location by specifying the memory address as described in the preceding paragraphs. After you specify the memory address, press the ALTER key. This will cause PAM-8 to enter the memory alter mode.

When PAM-8 enters the memory alter mode, a single decimal point rotates from right to left through all 9 digits. You can now alter the contents of the displayed location by entering the new octal value (three digits on the keypad). When the three digits have been entered, acoustical verification (a short beep) is given **and the memory address is incremented**. You can then alter this new location by entering three more digits or pressing one of the following keys, causing the monitor to perform the indicated function:

<u>KEY</u>	<u>FUNCTION</u>
+	Increment the address.
-	Decrement the address.
MEM	Specify a new memory address (leave memory alter mode).
REG	Specify a register for display (leave memory alter mode).
ALTER	Exit from the alter mode (into the display mode).

NOTE: PAM-8 automatically increments the memory address as each entry (3 octal digits) is complete. Therefore, you may load a program in sequential locations very rapidly. Each location is modified by simply entering the three octal digits.

The following example reviews each step as the H8 is turned on; the memory address mode is entered; and the location 040 123 is addressed, altered to 345, checked, and closed.

<u>DISPLAY</u>			<u>COMMENTS</u>
X X X	X X X	X X X	Random memory display at power up (X=random number.)
X.X.X.	X.X.X.	X.X.X.	MEM key pressed. (In memory address mode, a short beep.)
X.X.0.	X.X.X.	X.X.X.	0 key pressed. (Short beep.)
X.0.4.	X.X.X.	X.X.X.	4 key pressed. (Short beep.)
0.4.0.	X.X.X.	X.X.X.	0 key pressed. (Medium beep.) Contents of location 040 XXX displayed.)
0.4.0.	X.X.1.	X.X.X.	1 key pressed. (Short beep. Contents of 040 XX1 displayed.)
0.4.0.	X.1.2.	X.X.X.	2 key pressed. (Short beep. Contents of 040 X12 displayed.)
0 4 0	1 2 3	X X X	3 key pressed. (Medium beep. Contents of desired location 040 123 displayed, decimal points out.)
0.4.0	1.2.3	X.X.X	ALTER key pressed. (Short beep. Decimal points rotate .)
0.4.0.	1.2.3.	X.X.3.	3 key pressed. (Short beep. Decimal points rotate .)
0.4.0.	1.2.3.	X.3.4.	4 key pressed. (Short beep. Decimal points rotate .)
0.4.0.	1.2.4.	X.X.X.	5 key pressed. (Medium beep. Address increments one location. Decimal points rotate .)
0.4.0	1.2.3	3.4.5	–key pressed. (Short beep. Address decrements one location. Decimal points rotate .)
0 4 0	1 2 3	3 4 5	ALTER key pressed. (Short beep. Decimal points go out.)

Stepping Through Memory

When PAM-8 is either in the display memory or alter memory modes, the + and – keys increment and decrement the memory address. Each time you press the key, PAM-8 increments (or decrements) the memory address one location. If you hold the key down, the auto-repeat function of PAM-8 causes the memory address to increment or decrement repeatedly (approximately one location every second).

DISPLAYING AND ALTERING REGISTERS

PAM-8 can display and alter the contents of the 8080 CPU registers, just as it displays and alters the contents of H8 memory locations. Although the process is quite similar, a few special features should be noted.

Specifying a Register for Display

Press the REG key to specify that a register is to be displayed. After you press the REG key, press a second key (SP through PC, see the Table below) to specify the desired register or register pair.

When the REG key is pressed, six decimal points light, indicating that you must now select a register. NOTE: Simply pressing the REG key causes a register name to appear in the right-hand digits. However, you must select a register using the Register Select key before a register is definitely selected and its true contents are displayed. Once a register is selected, the decimal points are extinguished.

The contents of the selected register pair are displayed in the six left-most displays. The register name (or names) are displayed in the two right-most digits of the right-hand three displays. The registers are selected and displayed in accordance with the following table:

<u>KEY</u>	<u>LEFT 3 DIGITS</u>	<u>MIDDLE 3 DIGITS</u>	<u>RIGHT PAIR</u>	<u>COMMENTS</u>
SP (1)	000 to 377	000 to 377	<i>SP</i>	Stack pointer
AF (2)	000 to 377	000 to 377	<i>AF</i>	AF Register pair
BC (3)	000 to 377	000 to 377	<i>BC</i>	BC Register pair
DE (4)	000 to 377	000 to 377	<i>DE</i>	DE Register pair
HL (5)	000 to 377	000 to 377	<i>HL</i>	HL Register pair
PC (6)	000 to 377	000 to 377	<i>PC</i>	Program counter

NOTE: The contents of any single eight-bit register may lie in the range of 000 to 377 octal. The stack pointer (SP) and the program counter (PC) are 16-bit registers and are displayed as two sets of three octal numbers. Each 3-digit grouping corresponds to one byte (8 bit number). When a register pair is displayed, the left three digits correspond to the left register and the middle three digits correspond to the right register. For example:

256 312 AF

Register A contains 256 and F contains 312.

Altering the Contents of a Selected Register

To alter the contents of a register (or register pair), you must first specify it as described in the preceding paragraphs. After you select the register or register pair, press the ALTER key. This will cause the six left-hand decimal points to rotate right to left, indicating that you may enter 6 digits to alter the contents of the indicated register or register pair.

Alternatively, you may press one of the following command keys:

<u>KEY</u>	<u>FUNCTION</u>
+	Changes the register pair being displayed.
-	Changes the register pair being displayed.
MEM	Specify a new memory address (leave the alter register mode).
REG	Specify a new register for display (leave alter register mode).
ALTER	Exit the register alter mode.

NOTE: Stack pointer register (SP) is not a direct display of the real stack pointer register, but simply a copy of the real stack pointer register and is used for display purposes only. The stack pointer cannot be altered from the front panel. To alter the stack pointer register, an SPHL (SPHL = 371) instruction must be written into memory. The desired new stack pointer value is then placed in the HL register pair. PAM-8's single instruction mode is used to execute the SPHL swap instructions, loading the stack pointer with the contents loaded in the HL register pair.

Stepping Through the Registers

Use + and - keys to change the register pair being displayed. For example, if the DE register pair is being displayed, press the + key causes the next sequential register pair to be displayed (the HL pair). In the same manner, pressing the - key causes the register to decrement to the preceding pair. For example, if the DE pair is being displayed, pressing the - key displays the BC register pair. NOTE: Holding down either the + key or the - key causes the display to continuously increment or decrement through all the six registers/register pairs.



PROGRAM EXECUTION CONTROL

PAM-8 supports three basic program execution control facilities:

- Beginning or starting execution.
- Breakpointing.
- Single instruction.

Each of these execution controls permits the programmer to execute the desired portions of a program and examine its effects. He may execute the entire program, or a small group of instructions, or a single program instruction.

Initiating Program Execution

To begin the execution of a program residing in H8 memory, place the address of the first instruction to be executed in the PC (program counter). Use the methods described in "Displaying and Altering Registers" (Page 1-14). Once the address of this first instruction is placed in the program counter, press the GO key and program execution will begin. NOTE: Unless the program disables the front panel, the display continues to be actively updated, although the front panel commands are no longer active (except for RST and RTM). If the program counter is displayed when you press the GO key, PAM-8 continuously monitors the program counter.

Breakpointing

Breakpointing permits the programmer to execute small portions of a program and then return to PAM-8. Breakpointing is especially useful when a program is being "debugged." Small portions of the program may be executed and their results observed. If there is an error, it may be corrected before an entire program is involved.

When the H8 executes a program and encounters a halt instruction, it re-enters PAM-8 and sounds the alarm. All of the registers are preserved and the program counter points to the address **following** the address of the halt instruction. Thus, you can breakpoint a program from the front panel by inserting halt instructions (HLT = 166) at the desired points throughout the program. When a particular

section of the program is tested and the breakpoint feature is no longer required, you can change the halt to a NOP (NOP = 000). Once the halts are changed to NOPs, execution of the NOP simply passes control to the next successive instruction. Program execution for breakpointing uses the GO key as described above.

NOTE: If you temporarily replace an existing instruction with a halt, you must restore the instruction before resuming program execution. The contents of the program counter point to the address **following** the halt. Therefore, if the instruction which replaced the halt is to be executed, when the program continues, the contents of the program counter must be decremented one location before execution is resumed.

Single Instruction Operation

Any user program may be operated in the single instruction mode. This procedure is identical to the GO command, except that the SI key is pressed rather than the GO key. When the SI key is pressed, a single **instruction** (not a single machine cycle) is executed and then control is returned to PAM-8. Single instruction operation is available for careful inspection of program results and for executing special programs, such as swapping the HL register pair with the stack pointer as discussed in "Altering the Contents of a Selected Register" (Page 1-15).

Interrupting a Program During Execution

You can interrupt a running program (with all registers preserved at the point of interruption) by pressing RTM & 0. You can then examine and/or alter the contents of various memory locations and all the registers as required. Resume execution of the program at the next sequential instruction by simply pressing the GO key. NOTE: Although all registers and memory locations are preserved when RTM & 0 are pressed, it is very difficult to stop a program at an exact location. Therefore, use the breakpoint feature if you want to stop the program at an exact location.



LOAD/DUMP ROUTINES

PAM-8 contains a routine that lets you load and dump memory contents from or to a tape. This feature is especially important, as most computers require one or two successive “boot strap” routines to be hand-loaded before a desired program can be loaded into the main memory. All these “boot strap” routines are contained within the PAM-8 ROM, and use sophisticated error checking techniques. Thus, a program can be loaded or dumped by simply pressing a single key.

Loading From Tape

To load from a tape, ready the reader device with the tape to be loaded prior to executing the load command. Place PAM-8 in the display memory mode and press the LOAD key. Once the LOAD key is pressed, PAM-8 starts the tape transport and scans the tape for the first file record.

No change will be seen on the front panel displays until PAM-8 finds the first file. When the first file record is located, PAM-8 checks it to see if it is the first (or only) record in a sequence, and the record is a memory dump record. If it is not a memory dump record, a number two error is flagged (see “Tape Errors” on Page 1-20).

Once a correct record is found, loading proceeds. The loading procedure places the entry point address of the program being loaded in the H8 program counter. The H8 memory is then loaded. The displays continuously show the address being loaded and the data being loaded at these addresses. When the load is complete, PAM-8 sounds a long beep and displays the final memory address. If the load is faulty, a number one error is displayed and the audio alert continuously beeps. (See “Tape Errors,” Page 1-20.)

NOTE: You may abort a partial load by using the CANCEL key. Naturally, the load image resulting from this action is incorrect, and should not be executed.

Dumping to Tape

Before dumping a memory image onto tape, the following three dump parameters are required:

- The entry point address (the program starting address).
- The dump starting address.
- The dump ending address.

Set the desired entry point address by placing this value in the program counter (PC). This value will be placed in the program counter whenever you load the program so execution will begin at this address when you press the GO key.

Place the dump starting address into the first two H8 RAM cells. These are: 040 000 (offset octal) and 040 001 (offset octal). NOTE: The low order byte of the address should be placed into location 040 000 and the high order byte of the starting address should be placed into location 040 001.

Enter the dump ending address as a memory address using the # (MEM) key. Then ready the tape transport and press the DUMP key. As the tape dump takes place, the number of bytes left to be dumped and the contents of the memory location being dumped are displayed on the front panel. You can abort a dump by using the CANCEL key. If the CANCEL key is used, an incomplete dump image is left on the tape. This cannot be loaded at a future date. NOTE: A successful load automatically sets up the following three dump parameters:

- A. The program starting locations are stored in locations 040 000 and 040 001.
- B. The program ending location is displayed.
- C. The program counter contains the program entry point.

Figure 1-3A shows the steps of a typical dump sequence and Figure 1-3B shows the steps of a typical load sequence.

1. Set PC to 040 100; (040 100 = entry address).
2. Set 040 000 to 100 (100 = low byte of dump start).
3. Set 040 001 to 040 (040 = high byte of dump start).
4. Enter memory address 052 340 (052 340 = end address of dump).
5. Be sure tape is ready.
6. Press DUMP.

Figure 1-3A

The H8 memory image dump.

1. Be sure tape is ready.
2. Press LOAD.

Figure 1-3B

The H8 memory image load.



Copying a Tape

The beginning and final address of the load image are placed at the appropriate points. Thus, to copy a tape, simply load the tape as described in "Loading From Tape" (Page 1-18). Then ready the dump tape drive and press the DUMP key. A dump then takes place, including entry point, initial address, and final address.

In a similar manner, to load, alter, and then dump, enter only the ending address. The other parameters are unchanged from the load if locations 040 000, 040 001 or the program counter have not been modified during the altering procedure.

Tape Errors

PAM-8 detects two types of tape errors: record errors and checksum errors. In either case, when an error is detected, the tape transport is halted. The error number is then displayed in the center three digits (001 for a checksum error, 002 for a record error) and the alarm is repeatedly sounded. To halt the alarm and return to the command mode, press the CANCEL key.

RECORD ERRORS

The following are typical causes of record errors.

- Attempting to load a file which is not a memory image. For example, loading an editor text file or a BASIC program file.
- Attempting to start a load in the middle of a load image. Therefore missing the initialization information at the start of the file.
- A tape error which causes a portion of the load image to be missed so the next record read is not in the proper sequence.

CHECKSUM ERRORS

A checksum error is flagged when the CRC (Cyclical Redundancy Check) checksum following a record does not match the CRC calculated by PAM-8. This error means that the record is either incorrectly recorded or the load is faulty. In either case, the load should be attempted again. If successive loads result in repeated failures, the original tape must be suspected as faulty.

I/O FACILITIES

PAM-8 supports two commands that allow you to perform input and output functions on H8 I/O ports. These front panel instructions permit simple manipulation of the H8 I/O ports without your having to write extensive routines to perform these functions.

Inputting From a Port

To input from a port, press the # key. Then enter three zero digits and the three-digit address (octal) of the desired port. NOTE: The front panel should now display 000 AAA, where AAA is the port address and 000 is meaningless. Press the IN key to read the port, the value is displayed in the three left-most digits of the front panel display.

Outputting to a Port

To output to a specified port, press the # key. Then enter the value to be supplied to the port in the three left-most displays. The port address is entered into the middle three displays. The display is of the form VVV AAA, where V stands for value, and A for address. Pressing the OUT key causes the value to be outputted to the indicated port.

Addressing Port Pairs

Frequently, ports are assigned in pairs, where one of the two port addresses is the control and status register and the other port is the data port. Address port pairs by using the + and - key to change ports. Once the initial port has been defined, the + key increments the port address to a new higher numbered port, and the - key is used to decrement to a lower numbered port.



ADVANCED CONTROL

One of the advanced features of PAM-8 is its provisions allowing sophisticated users to augment or replace PAM-8's functions. Augmenting or replacing PAM-8 functions is usually done in conjunction with assembly language programs. Sometimes it is possible to implement these features by using the POKE and PEEK commands in BASIC. The sample exercise in "Appendix B" (Page 1-64) uses several PAM-8 functions, including the clock, I/O, and the audio alarm.

The following discussion refers to symbols and locations defined in the PAM-8 program listing, given in its complete form as "Appendix A." It is recommended that you review the PAM-8 listing in order to become familiar with its various features. This can be done in conjunction with reading the following section, or independently. In either case, a first overview followed by a detailed analysis of the listing is probably necessary for a complete understanding.

16-Bit Tick Counter (TICCNT)

PAM-8 maintains a 16-bit (2 byte) tick counter known as TICCNT. The value of this counter is incremented each time a clock interrupt is processed. As an interrupt occurs once every 2 mS, the counter is incremented once every 2 mS. As long as clock interrupts are not disabled, this value can be used by any program to compute elapsed time. The tick counter may be set to any desired value, but it should not be frequently reset, as this interferes with the front panel refresh cycle. The contents of the tick counter are contained in memory locations 040 033 (the least significant byte) and 040 034 (the most significant byte).

Using the Keypad

When your program is running, PAM-8 does not recognize any single key command. Thus, all single key patterns are available for your program. To read keypad patterns, you can use one of two routines. First, you may take an input from port IP. PAD; or second, your program may use PAM-8's RCK routine. The input port IP. PAD is permanently assigned to port location 360. Inputting a binary number from this port detects which of the 16 keys are depressed. These results are shown in the table on Page 1-57 of "Appendix A."

A far more sophisticated keypad routine is available to you in the RCK (read Console Keypad) routine. This is also described in "Appendix A" (see Page 1-57). RCK provides keypad decoding, keypad debounce routines, auto-repeat routines, and acoustical feedback.

NOTE: If you use two key combinations, each key must reside in a separate bank. The first bank includes keys 0-7 and the second bank includes keys 8-#. RCK cannot decode two key combinations.

Display Usage

When a user program is running, PAM-8 normally displays the contents of the selected register or memory location. However, you may disable this process and display any arbitrary segment pattern, or completely disable the display to provide greater computational through-put. The display usage is primarily controlled by setting various bits in the .MFLAG memory cell. This memory cell is found at location 040 010.

MANUAL UPDATING

By setting the UO.DDU (see "Appendix A," Page 1-29, for an explanation of the user option bits, UO.XXX) bit in the .MFLAG memory location, you can instruct PAM-8 to continue refreshing the front panel displays but to disable updating. When this is done, PAM-8 continues to refresh the LED's from a 9-byte block of RAM cells found at locations 040 013 through 040 023. A description of these front panel LED's (FPLEDS) is found in "Appendix A" (see Page 1-60). When the UO.DDU bit is set in .MFLAG, the contents of these bytes are not altered in any manner by PAM-8.

You can use this technique to display numbers, letters, or arbitrary bar patterns (see Page 1-58) on the front panel displays. For instance, your program may alter the display by inserting any value into FPLEDS. The front panel LED segments will display a decimal integer if you use the octal to 7-segment pattern (DODA) display.

MANUAL DISPLAY REFRESHING

By setting the UO.NFR (User Option.No Front Panel Refresh) bit in the .MFLAG memory cell, you can instruct PAM-8 to stop refreshing the front panel displays. Setting the UO.NFR bit does not disable the clock interrupts; therefore, the tick counter (TICCNT) is still incremented. But PAM-8 does not refresh the displays from the information contained in the FPLEDS bytes.

NOTE: If you desire, you may write a program to refresh the front panel LED displays. Usually this is done using the clock interrupts. If you undertake an independent front panel refresh program, take extreme care to avoid burning the displays due to excessive refreshing. **The total power dissipated in the LEDs is determined by the refresh cycle, and too frequent refreshing will result in excessive display heating.**

Using Interrupts

All H-8 interrupts cause control to be transferred into the low 64 bytes of memory. PAM-8 occupies this memory space so all interrupts are first processed by PAM-8. Except for level zero interrupts, which are used as master clears, you can supply an interrupt processing routine for each of the seven additional interrupts. The following sections explain the use of each of these interrupts.

I/O INTERRUPTS

Interrupts numbered 3 through 7 are I/O interrupts. PAM-8 does not process these interrupts in any way. When a level 3 through level 7 interrupt is received, PAM-8 immediately transfers to the user interrupt vectors contained in memory locations 040 037 through 040 064. These locations are listed in "Appendix A" (see Page 1-61). Each location must contain a jump instruction pointing to the appropriate program location which processes these interrupts.

NOTE: If any of these interrupts occur, you must supply a processing routine for them. This routine must be complete including both entry and exit processing. When you use H8 interrupts, you must use only the available vector which is 6 to insure compatability with future H8 products. You may also use 2 if you will not be using BUG-8.

CLOCK INTERRUPTS

The level one interrupts are generated by the front panel hardware every 2 mS. PAM-8 normally processes these interrupts. However, by setting a processing vector in UIVC and setting the UO.INT bit in the MFLAG cell, PAM-8 enters the users routine each time a clock interrupt is generated. "Appendix A" (see Page 1-31) gives the required entry and exit conditions for processing clock interrupts.

SINGLE INSTRUCTION AND BREAKPOINT INTERRUPTS

Level two interrupts are generated by the single instruction hardware contained on the CPU card. When a single instruction is requested, the result of the interrupt is processed by PAM-8. If the single instruction interrupt was generated by PAM-8 in response to a Monitor Mode Single Instruction register condition, PAM-8 processes it. Otherwise, PAM-8 jumps to the user level two interrupt vector (UIVC). Since the level two interrupt does not affect PAM-8, a level two restart instruction can be used as a breakpoint instruction by the user programs.

APPENDIX A

This appendix contains a complete listing of the PAM-8 front panel monitor program. PAM-8 resides in the low 1,024 bytes of the H8 computer. It provides all the control for front panel operation, and cassette or paper tape load and dump facilities. It also provides for master clear and front panel interrupt processing. PAM-8 presumes RAM cells are available for its use in locations 040 000 through 040 077 and 80 bytes are available in high memory for a stack. The use of these RAM cells is described on Page 1-60 of this Appendix and in the memory map on Page 0-47.

Pages 1-61, 1-62, and 1-63 of this Appendix are a symbolic reference table. Use this table to find the program locations where each symbolic address is used. Symbolic addresses are listed in alphabetical sequence.

PAM/8 - H8 FRONT PANEL MONITOR. #01.00.00.
INTRODUCTION.

HEATH X8ASM V1.1 06/21/77
15:43:50 01-APR-77 PAGE 1

```

4 *** PAM/8 - H8 FRONT PANEL MONITOR.
5 *
6 * JGL, 05/01/76,
7 *
8 * FOR *WINTER* INC.
9 *
10 * COPYRIGHT 05/1976, WINTER CORPORATION,
11 * 902 N. 9TH ST.
12 * LAFAYETTE, IND.

```

```

14 *** PAM/8 - H8 FRONT PANEL MONITOR.
15 *
16 * THIS PROGRAM RESIDES (IN ROM) IN THE LOW 1024 BYTES OF THE HEATH
17 * H8 COMPUTER. IT ACTUALLY CONSISTS OF TWO VIRTUALLY INDEPENDENT
18 * ROUTINES: A TASK-TIME PROGRAM WHICH PROVIDES SOPHISTICATED
19 * FRONT PANEL MONITOR SERVICE, AND AN INTERRUPT-TIME PROGRAM WHICH
20 * PROVIDES BOTH A REAL-TIME CLOCK AND EMULATES AN EFFECTIVE
21 * HARDWARE FRONT PANEL.

```

```

23 *** INTERRUPTS.
24 *
25 * PAM/8 IS THE PRIMARY PROCESSOR FOR ALL INTERRUPTS.
26 * THEY ARE PROCESSED AS FOLLOWS:
27 *
28 * RST USE
29 *
30 * 0 MASTER CLEAR. (NEVER USED FOR I/O OR RST)
31 *
32 * 1 CLOCK INTERRUPT. NORMALLY TAKEN BY PAM/8,
33 * SETTING BIT *UO.CLK* IN BYTE *MFLAG* ALLOWS
34 * USER PROCESSING (VIA A JUMP THROUGH *UIVEC*),
35 * UPON ENTRY OF THE USER ROUTINE, THE STACK
36 * CONTAINS:
37 * (STACK+0) = RETURN ADDRESS (TO PAM/8)
38 * (STACK+2) = (STACKPTR+14)
39 * (STACK+4) = (AF)
40 * (STACK+6) = (BC)
41 * (STACK+8) = (DE)
42 * (STACK+10) = (HL)
43 * (STACK+12) = (PC)
44 * THE USER'S ROUTINE SHOULD RETURN TO PAM/8 VIA
45 * A *RET* WITHOUT ENABLING INTERRUPTS.
46 *
47 * 2 SINGLE STEP. SINGLE STEP INTERRUPTS GENERATED
48 * BY PAM/8 ARE PROCESSED BY PAM/8.
49 * ANY SINGLE STEP INTERRUPT RECEIVED WHEN IN
50 * USER MODE CAUSES A JUMP THROUGH *UIVEC*+3.
51 * STACK UPON USER ROUTINE ENTRY:
52 * (STACK+0) = (STACKPTR+12)
53 * (STACK+2) = (AF)
54 * (STACK+4) = (BC)

```

```
55 *      (STACK+6) = (DE)
56 *      (STACK+8) = (HL)
57 *      (STACK+10) = (PC)
58 *      THE USER'S ROUTINE SHOULD HANDLE ITS OWN RETURN
59 *      FROM THE INTERRUPT.
60 *
61 *
62 *      THE FOLLOWING INTERRUPTS ARE VECTORED DIRECTLY THROUGH *UIVEC*.
63 *      THE USER ROUTINE MUST HAVE SETUP A JUMP IN *UIVEC* BEFORE ANY
64 *      OF THESE INTERRUPTS MAY OCCUR.
65 *
66 *      3      I/O 3. CAUSES A DIRECT JUMP THROUGH *UIVEC*+6
67 *
68 *      4      I/O 4. CAUSES A DIRECT JUMP THROUGH *UIVEC*+9
69 *
70 *      5      I/O 5. CAUSES A DIRECT JUMP THROUGH *UIVEC*+12
71 *
72 *      6      I/O 6. CAUSES A DIRECT JUMP THROUGH *UIVEC*+15
73 *
74 *      7      I/O 7. CAUSES A DIRECT JUMP THROUGH *UIVEC*+18
```

PAM/8 - HB FRONT PANEL MONITOR #01.00.00.

HEATH XBASM V1.1 06/21/77

ASSEMBLY CONSTANTS.

15:43:52 01-APR-77 PAGE 3

77 ** ASSEMBLY CONSTANTS

79 ** I/O PORTS

000.360	81	IP.PAD	EQU	360Q	PAD INPUT PORT
000.360	82	OP.CTL	EQU	360Q	CONTROL OUTPUT PORT
000.360	83	OP.DIG	EQU	360Q	DIGIT SELECT OUTPUT PORT
000.361	84	OP.SEG	EQU	361Q	SEGMENT SELECT OUTPUT PORT
000.371	85	IP.TFC	EQU	371Q	TAPE CONTROL IN
000.371	86	OP.TFC	EQU	371Q	TAPE CONTROL OUT
000.370	87	IP.TFD	EQU	370Q	TAPE DATA IN
000.370	88	OP.TFD	EQU	370Q	TAPE DATA OUT

90 ** ASCII CHARACTERS.

000.026	92	A.SYN	EQU	026Q	SYNC CHARACTER
000.002	93	A.STX	EQU	002Q	STX CHARACTER

95 ** FRONT PANEL HARDWARE CONTROL BITS.

000.020	97	CB.SSI	EQU	00010000B	SINGLE STEP INTERRUPT
000.040	98	CB.MTL	EQU	00100000B	MONITOR LIGHT
000.100	99	CB.CLI	EQU	01000000B	CLOCK INTERRUPT ENABLE
000.200	100	CB.SPK	EQU	10000000B	SPEAKER ENABLE

102 ** DISPLAY MODE FLAGS (IN *DSPMOD*)

000.000	104	DM.MR	EQU	0	MEMORY READ
000.001	105	DM.MW	EQU	1	MEMORY WRITE
000.002	106	DM.RR	EQU	2	REGISTER READ
000.003	107	DM.RW	EQU	3	REGISTER WRITE
000.000	108	XTEXT	TAPE		TAPE DEFINITIONS

110X ** TAPE EQUIVALENCES.

000.001	112X	RT.MI	EQU	1	RECORD TYPE - MEMORY DUMP IMAGE
000.002	113X	RT.BP	EQU	2	RECORD TYPE - BASIC PROGRAM
000.003	114X	RT.CT	EQU	3	RECORD TYPE - COMPRESSED TEXT

116X ** BLOCK SIZE FOR INTER-PRODUCT COMMUNICATION.

002.000	118X	BLKSIZ	EQU	512	
---------	------	--------	-----	-----	--

PAM/8 - H8 FRONT PANEL MONITOR #01.00.00.
ASSEMBLY CONSTANTS.

HEATH XBASM V1.1 06/21/77
15:43:56 01-APR-77 PAGE 4

```
121 ** MACHINE INSTRUCTIONS.
122
000.166 123 MI.HLT EQU 01110110B HALT
000.311 124 MI.RET EQU 11001001B RETURN
000.333 125 MI.IN EQU 11011011B INPUT
000.323 126 MI.OUT EQU 11010011B OUTPUT
000.072 127 MI.LDA EQU 00111010B LDA
000.346 128 MI.ANI EQU 11100110B ANI
000.021 129 MI.LXID EQU 00010001B LXI D

131 ** USER OPTION BITS.
132 *
133 * THESE BITS ARE SET IN CELL .MFLAG. 040 010
134
000.200 135 UD.HLT EQU 10000000B DISABLE HALT PROCESSING
000.100 136 UD.NFR EQU CB.CLI NO REFRESH OF FRONT PANEL
000.002 137 UD.DDU EQU 00000010B DISABLE DISPLAY UPDATE
000.001 138 UD.CLK EQU 00000001B ALLOW CLOCK INTERRUPT PROCESSING

000.000 140 XTEXT U8251 DEFINE 8251 USART BITS
```

PAM/8 - HB FRONT PANEL MONITOR \$01.00.00.
8251 USART BIT DEFINITIONS.

HEATH X8ASM V1.0 02/18/77
13:23:23 01-APR-77 PAGE 5

```

143X **      8251 USART BIT DEFINITIONS.
144X *
145X
146X **      MODE INSTRUCTION CONTROL BITS.
147X
000.100      148X UMI.1B EQU 01000000B 1 STOP BIT
000.200      149X UMI.HB EQU 10000000B 1 1/2 STOP BITS
000.300      150X UMI.2B EQU 11000000B 2 STOP BITS
000.040      151X UMI.PE EQU 00100000B EVEN PARITY
000.020      152X UMI.PA EQU 00010000B USE PARITY
000.000      153X UMI.L5 EQU 00000000B 5 BIT CHARACTERS
000.004      154X UMI.L6 EQU 00000100B 6 BIT CHARACTERS
000.010      155X UMI.L7 EQU 00001000B 7 BIT CHARACTERS
000.014      156X UMI.L8 EQU 00001100B 8 BIT CHARACTERS
000.001      157X UMI.1X EQU 00000001B CLOCK X 1
000.002      158X UMI.16X EQU 00000010B CLOCK X 16
000.003      159X UMI.64X EQU 00000011B CLOCK X 64
160X
161X **      COMMAND INSTRUCTION BITS.
162X
000.100      163X UCI.IR EQU 01000000B INTERNAL RESET
000.040      164X UCI.R0 EQU 00100000B READER-ON CONTROL FLAG
000.020      165X UCI.ER EQU 00010000B ERROR RESET
000.004      166X UCI.RE EQU 00000100B RECEIVE ENABLE
000.002      167X UCI.IE EQU 00000010B ENABLE INTERRUPTS FLAG
000.001      168X UCI.TE EQU 00000001B TRANSMIT ENABLE
169X
170X **      STATUS READ COMMAND BITS.
171X
000.040      172X USR.FE EQU 00100000B FRAMING ERROR
000.020      173X USR.OE EQU 00010000B OVERRUN ERROR
000.010      174X USR.PE EQU 00001000B PARITY ERROR
000.004      175X USR.TXE EQU 00000100B TRANSMITTER EMPTY
000.002      176X USR.RXR EQU 00000010B RECEIVER READY
000.001      177X USR.TXR EQU 00000001B TRANSMITTER READY

```



```

180 ***   INTERRUPT VECTORS.
181 *
182
184 **   LEVEL 0 - RESET
185 *
186 *   THIS 'INTERRUPT' MAY NOT BE PROCESSED BY A USER PROGRAM.
187
000.000 188   ORG      00A
189
000.000 021 371 003 190 INIT0  LXI    D,PRSR0M      (DE) = ROM COPY OF PRS CODE
000.003 041 012 040 191   LXI    H,PRSRAM+PRSL-1 (HL) = RAM DESTINATION FOR CODE
000.006 303 073 000 192   JMP     INIT      INITIALIZE
377.073 193   ERRPL  INIT-1000A    BYTE IN WORD 10A MUST BE 0

195 **   LEVEL 1 - CLOCK
196
000.010 197 INT1   EQU     100      INTERRUPT ENTRY POINT
198
000.000 199   ERRNZ  *-110      INTO TAKES UP ONE BYTE
000.011 315 132 000 200   CALL  SAVALL    SAVE USER REGISTERS
000.014 026 000 201   MVI    D,0
000.016 303 201 000 202   JMP     CLOCK    PROCESS CLOCK INTERRUPT
377.201 203   ERRPL  CLOCK-1000A   EXTRA BYTE MUST BE 0

205 **   LEVEL 2 - SINGLE STEP
206 *
207 *   IF THIS INTERRUPT IS RECEIVED WHEN NOT IN MONITOR MODE,
208 *   THEN IT IS ASSUMED TO BE GENERATED BY A USER PROGRAM
209 *   (SINGLE STEPPING OR BREAKPOINTING). IN SUCH CASE, THE
210 *   USER PROGRAM IS ENTERED THROUGH (UIVEC+3)
211
000.020 212 INT2   EQU     20A      LEVEL 2 ENTRY
213
000.000 214   ERRNZ  *-21A      INT1 TAKES EXTRA BYTE
000.021 315 132 000 215   CALL  SAVALL    SAVE REGISTERS
000.024 032 216   LDAX   D      (A) = (CTLFLG)
040.011 217   SET    CTLFLG
000.025 303 244 001 218   JMP     STPRTN    STEP RETURN

220 ***   I/O INTERRUPT VECTORS.
221 *
222 *   INTERRUPTS 3 THROUGH 7 ARE AVAILABLE FOR GENERAL I/O USE.
223 *
224 *   THESE INTERRUPTS ARE NOT SUPPORTED BY PAM/8, AND SHOULD
225 *   NEVER OCCUR UNLESS THE USER HAS SUPPLIED HANDLER ROUTINES
226 *   (THROUGH UIVEC)
227

```

PAM/B - H8 FRONT PANEL MONITOR #01.00.00.
HARDWARE INTERRUPT VECTORS

HEATH X8ASM V1.0 02/18/77
13:23:26 01-APR-77 PAGE 7

```

000.030          228      ORG      30A
000.030  303 045 040  229  INT3   JMP      UIVEC+6      JUMP TO USER ROUTINE
                                230
000.033  064 064 064  231      DB      '44413'      HEATH PART NUMBER 444-13
                                232

000.040          233      ORG      40A
000.040  303 050 040  234  INT4   JMP      UIVEC+9      JUMP TO USER ROUTINE
                                235
000.043  100 112 107  236      DB      100R,112R,107R,114R,100R      SUPPORT CODE
                                237

000.050          238      ORG      50A
000.050  303 053 040  239  INT5   JMP      UIVEC+12     JUMP TO USER ROUTINE
                                240
                                241
                                242  **      DLY - DELAY TIME INTERVAL.
                                243  *
                                244  *      ENTRY  (A) = MILLISECOND DELAY COUNT/2
                                245  *      EXIT   NONE
                                246  *      USES   A,F
                                247
000.053  365          248  DLY     PUSH    PSW          SAVE COUNT
000.054  257          249      XRA     A              DONT SOUND HORN
000.055  303 143 002  250      JMP     HRNO          PROCESS AS HORN
                                251

000.060          252      ORG      60A
000.060  303 056 040  253  INT6   JMP      UIVEC+15     JUMP TO USER ROUTINE
                                254
                                255
000.063  076 320      256  GO.     MVI     A,CB,SSI+CB,CLI+CB,SPK  OFF MONITOR MODE LIGHT
000.065  303 235 001  257      JMP     SSI1          RETURN TO USER PROGRAM
                                258

000.070          259      ORG      70A
000.070  303 061 040  260  INT7   JMP      UIVEC+18     JUMP TO USER ROUTINE
                                261

```

PAM/8 - H8 FRONT PANEL MONITOR #01.00.00.
MASTER CLEAR PROCESSING

HEATH XBASH V1.0 02/18/77
13:23:28 01-APR-77 PAGE 8

```

263 **      INIT - INITIALIZE SYSTEM
264 *
265 *      INIT IS CALLED WHENEVER A HARDWARE MASTER-CLEAR IS INITIATED.
266 *
267 *      SETUP PAM/8 CONTROL CELLS IN RAM.
268 *      DECODE HOW MUCH MEMORY EXISTS, SETUP STACKPOINTER, AND
269 *      ENTER THE MONITOR LOOP.
270 *
271 *      ENTRY FROM MASTER CLEAR
272 *      EXIT INTO PAM/8 MAIN LOOP
273
274
000.073 032 275 INIT LDAX D COPY *PRSRM* INTO RAM
000.074 167 276 MOV M,A MOVE BYTE
000.075 053 277 DCX H DECREMENT DESTINATION
000.076 034 278 INR E INCREMENT SOURCE
000.077 302 073 000 279 JNZ INIT IF NOT DONE
280
004.000 281 SINCX EQU 4000A SEARCH INCREMENT
282
000.102 026 004 283 MVI D,SINCX/256 (DE) = SEARCH INCREMENT
000.104 041 000 034 284 LXI H,START-SINCX (HL) = FIRST RAM - SEARCH INCREMENT
285
286 *      DETERMINE MEMORY LIMIT.
287
000.107 167 288 INIT1 MOV M,A RESTORE VALUE READ
000.110 031 289 DAD D INCREMENT TRIAL ADDRESS
000.111 176 290 MOV A,M (A) = CURRENT MEMORY VALUE
000.112 065 291 DCR M TRY TO CHANGE IT
000.113 276 292 CMP M
000.114 302 107 000 293 JNE INIT1 IF MEMORY CHANGED
294
000.117 053 295 INIT2 DCX H
000.120 371 296 SPHL SET STACKPOINTER = MEMORY LIMIT -1
000.121 345 297 PUSH H SET *PC* VALUE ON STACK
000.122 041 322 000 298 LXI H,ERROR
000.125 345 299 PUSH H SET 'RETURN ADDRESS'
300
301 *      CONFIGURE LOAD/DUMP UART
302
000.126 076 116 303 MVI A,UMI.1B+UMI.LB+UMI.16X
000.130 323 371 304 OUT OP.TPC SET 8 BIT, NO PARITY, 1 STOP, x16

```

PAM/8 - H8 FRONT PANEL MONITOR \$01.00.00.

HEATH X8ASM V1.0 02/18/77

INTERRUPT TIME SUBROUTINES

13:23:29 01-APR-77 PAGE 9

```

307 **      SAVALL - SAVE ALL REGISTERS ON STACK.
308 *
309 *      SAVALL IS CALLED WHEN AN INTERRUPT IS ACCEPTED, IN ORDER TO
310 *      SAVE THE CONTENTS OF THE REGISTERS ON THE STACK.
311 *
312 *      ENTRY CALLED DIRECTLY FROM INTERRUPT ROUTINE.
313 *      EXIT  ALL REGISTERS PUSHED ON STACK,
314 *      IF NOT YET IN MONITOR MODE, REGPTR = ADDRESS OF REGISTERS
315 *      ON STACK.
316 *      (DE) = ADDRESS OF CTLFLG
317
318
319 SAVALL    XTHL      SET H,L ON STACK TOP
320          PUSH      D
321          PUSH      B
322          PUSH      PSW
323          XCHG      (D,E) = RETURN ADDRESS
324          LXI      H,10
325          DAD      SP      (H,L) = ADDRESS OF USERS SP
326          PUSH      H      SET ON STACK AS 'REGISTER'
327          PUSH      D      SET RETURN ADDRESS
328          LXI      D,CTLFLG
329          LDAX     D      (A) = CTLFLG
330          CMA
331          ANI      CB.MTL+CB.SSI  SAVE REGISTER ADDR IF USER OR SINGLE-STEP
332          RZ          RETURN IF WAS INTERRUPT OF MONITOR LOOP
333          LXI      H,2
334          DAD      SP      (H,L) = ADDRESS OF 'STACKPTR' ON STACK
335          SHLD     REGPTR
336          RET

338 **      CUI - CHECK FOR USER INTERRUPT PROCESSING.
339 *
340 *      CUI IS CALLED TO SEE IF THE USER HAS SPECIFIED PROCESSING
341 *      FOR THE CLOCK INTERRUPT.
342
343
344          SET      .MFLAG      REFERENCE TO MFLAG
345 CUI1      LDAX     B      (A) = .MFLAG
346          ERNZ     U0,CLK-1      CODE ASSUMED = 01
347          RRC
348          CC       UIVEC      IF SPECIFIED, TRANSFER TO USER
349
350 *      RETURN TO PROGRAM FROM INTERRUPT.
351
352 INTXIT    POP      PSW      REMOVE FAKE 'STACK REGISTER'
353          POP      PSW
354          POP      B
355          POP      D
356          POP      H
357          EI
358          RET
000.132 343
000.133 325
000.134 305
000.135 365
000.136 353
000.137 041 012 000
000.142 071
000.143 345
000.144 325
000.145 021 011 040
000.150 032
000.151 057
000.152 346 060
000.154 310
000.155 041 002 000
000.160 071
000.161 042 035 040
000.164 311
040.010
000.165 012
000.000
000.166 017
000.167 334 037 040
000.172 361
000.173 361
000.174 301
000.175 321
000.176 341
000.177 373
000.200 311

```

```

361 ***      CLOCK - PROCESS CLOCK INTERRUPT
362 *
363 *      CLOCK IS ENTERED WHENEVER A MILLISECOND CLOCK INTERRUPT IS
364 *      PROCESSED.
365 *
366 *      TICCNT IS INCREMENTED EVERY INTERRUPT.
367
368
000.201 052 033 040 369 CLOCK LHLI TICCNT
000.204 043          370      INX H
000.205 042 033 040 371      SHLI TICCNT      INCREMENT TICCOUNT
372
373 **      REFRESH FRONT PANEL.
374 *
375 *      THIS CODE DISPLAYS THE APPROPRIATE PATTERN ON THE
376 *      FRONT PANEL LEDS. THE LEDS ARE PAINTED IN REVERSE ORDER,
377 *      ONE PER INTERRUPT. FIRST, NUMBER 9 IS LIT, THEN NUMBER 8,
378 *      ETC.
379
380
000.210 041 010 040 381      LXI H,MFLAG
000.213 176          382      MOV A,M
000.214 107          383      MOV B,A      (B) = CURRENT FLAG
000.215 346 100      384      ANI UD.NFR      SEE IF FRONT PANEL REFRESH WANTED
000.217 043          385      INX H
000.000          386      ERRNZ CTLFLG-MFLAG-1
000.220 176          387      MOV A,M      (A) = CTLFLG
000.221 112          388      MOV C,D      (C) = 0 IN CASE NO PANEL DISPLAY
000.222 302 237 000 389      JNZ CLK3      IF NOT
000.225 043          390      INX H      (H,L) = (REFIND)
000.000          391      ERRNZ REFIND-CTLFLG-1
000.226 065          392      DCR M      DECREMENT DIGIT INDEX
000.227 302 234 000 393      JNZ CLK2      IF NOT WRAP-AROUND
000.232 066 011      394      MVI M,9      WRAP DISPLAY AROUND
000.234 136          395      MOV E,M
000.235 031          396      DAD D      (H,L) = ADDRESS OF PATTERN
000.236 113          397      MOV C,E
000.237          398      CLK3 EQU *      (A) = CTNLG
000.237 261          399      ORA C      (A) = INDEX + FIXED BITS
000.240 323 360      400      OUT OP.DIG      SELECT DIGIT
000.242 176          401      MOV A,M
000.243 323 361      402      OUT OP.SEG      SELECT SEGMENT
403
404 *      SEE IF TIME TO DECODE DISPLAY VALUES.
405
000.245 056 033      406      MVI L,*TICCNT
000.247 176          407      MOV A,M
000.250 346 037      408      ANI 370      EVERY 32 INTERRUPTS
000.252 314 161 003 409      CZ      UFD      UPDATE FRONT PANEL DISPLAYS
410
411 *      EXII CLOCK INTERRUPT.
412
000.255 001 011 040 413      LXI B,CTLFLG
000.260 012          414      LDAX B      (A) = CTLFLG
000.261 346 040      415      ANI CB.MTL
000.263 302 172 000 416      JNZ INTXIT      IF IN MONITOR MODE

```

PAM/8 - H8 FRONT PANEL MONITOR #01.00.00.
PROCESS CLOCK INTERRUPTS

HEATH X8ASM V1.0 02/18/77
13:23:34 01-APR-77 PAGE 11

```

000.266 013      417      DCX      R
000.000          418      ERRNZ   CTLFLG-,MFLAG-1
000.267 012      419      LDAX    R      (A) = ,MFLAG
000.000          420      ERRNZ   UD.HLT-2000      ASSUME HIGH-ORDER
000.270 027      421      RAL
000.271 332 313 000 422      JC      CLK4      SKIP IT
          423
          424 *      NOT IN MONITOR MODE. CHECK FOR HALT
          425
000.274 076 012  426      MVI      A,10      (A) = INDEX OF *P* REG
000.276 315 052 003 427      CALL   LRA,      LOCATE REGISTER ADDRESS
000.301 136      428      MOV      E,M
000.302 043      429      INX      H
000.303 126      430      MOV      D,M      (D,E) = PC CONTENTS
000.304 033      431      DCX      D
000.305 032      432      LDAX    D
000.306 376 166  433      CPI      MI,HLT      CHECK FOR HALT
000.310 312 322 000 434      JE      ERROR      IF HALT, RE IN MONITOR MODE
          435
          436 *      CHECK FOR 'RETURN TO MONITOR' KEY ENTRY.
          437
000.313          438 CLK4    EQU      *
000.313 333 360  439      IN      IP,PAD
000.315 376 056  440      CPI      560      SEE IF '0' AND *
000.317 302 165 000 441      JNE     CUI1      IF NOT, ALLOW USER PROCESSING OF CLOCK

```

```

445 ***      ERROR - COMMAND ERROR.
446 *
447 *      ERROR IS CALLED AS A 'BAIL-OUT' ROUTINE.
448 *
449 *      IT RESETS THE OPERATIONAL MODE, AND RESTORES THE STACKPOINTER.
450 *
451 *      ENTRY  NONE
452 *      EXIT   TO MTR LOOP
453 *      CTLFLG SET
454 *      MFLAG 'CLEARED'
455 *      USES   ALL
456
457
000.322      458 ERROR EQU *
000.322 041 010 040 459 LXI H,MFLAG
000.325 176 460 MOV A,M (A) = MFLAG
000.326 346 275 461 ANI 377Q-UO,IDU-UO,NFR RE-ENABLE DISPLAYS
000.330 167 462 MOV M,A REPLACE
000.331 043 463 INX H
000.332 066 360 464 MVI M,CB.SSI+CB.MTL+CB.CLI+CB.SPK RESTORE *CTLFLG*
000.000 465 ERNZ CTLFLG-MFLAG-1
000.334 373 466 EI
000.335 052 035 040 467 LHLD REGPTR
000.340 371 468 SPHL RESTORE STACK POINTER TO EMPTY STATE
000.341 315 136 002 469 CALL ALARM ALARM FOR 200 MS

```

```

471 **      MTR - MONITOR LOOP.
472 *
473 *      THIS IS THE MAIN EXECUTIVE LOOP FOR THE FRONT PANEL EMULATOR.
474
475
000.344      476 MTR EQU *
000.344 373 477 EI
478
000.345 041 345 000 479 MTR1 LXI H,MTR1
000.350 345 480 PUSH H SET 'MTR1' AS RETURN ADDRESS
000.351 001 007 040 481 LXI B,DSPMOD (BC) = #DSPMOD
000.354 012 482 LDAX B
000.355 346 001 483 ANI 1 (A) = 1 IF ALTER
000.357 057 484 CMA
000.360 062 006 040 485 STA DSFROT ROTATE LED PERIODS IF ALTER
486
487 *      READ KEY
488
000.363 315 260 003 489 CALL RCK READ CONSOLE KEYPAD
000.366 052 024 040 490 LHLD ARUSS
000.371 376 012 491 CPI 10
000.373 322 005 001 492 JNC MTR4 IF IN 'ALWAYS VALID' GROUP
000.376 137 493 MOV E,A SAVE VALUE
040.007 494 SET DSFMOD
000.377 012 495 LDAX B (A) = DSPMOD
001.000 017 496 RRC
001.001 332 051 001 497 JC MTR5 IF IN ALTER MODE

```

PAM/8 - H8 FRONT PANEL MONITOR #01.00.00.

HEATH X8ASM V1.0 02/18/77

MTR - MAIN EXECUTIVE LOOP.

13:23:37 01-APR-77 PAGE 13

```

001.004 173      498      MOV      A,E      (A) = CODE
                499
                500 *      HAVE A COMMAND (NOT A VALUE)
                501
001.005 326 004  502 MTR4  SUI      4      (A) = COMMAND
001.007 332 322 000 503      JC      ERROR      IF BAD
001.012 137      504      MOV      E,A
001.013 345      505      PUSH     H      SAVE ABUSS VALUE
001.014 041 035 001 506      LXI     H,MTRA
001.017 026 000  507      MVI     D,0
001.021 031      508      DAD      D      (H,L) = ADDRESS OF TABLE ENTRY
001.022 136      509      MOV      E,m
001.023 031      510      DAD      D      (H,L) = ADDRESS OF PROCESSOR
001.024 343      511      XTHL     (SET ADDRESS, (H,L) = (ABUSS))
001.025 021 005 040 512      LXI     D,REGI      (D,E) = ADDRESS OF REG. INDEX
040.007          513      SET      DSPMOD
001.030 012      514      LDAX     R      (A) = DSPMOD
001.031 346 002  515      ANI     2      SET 'Z' IF MEMORY
001.033 012      516      LDAX     R      (A) = DSPMOD
001.034 311      517      RET      JUMP TO PROCESSOR
                518
                519
001.035          520 MTRA  EQU      *      JUMP TABLE
001.036 165      521      DB      GO-*      4 - GO
001.036 141      522      DB      IN-*      5 - INPUT
001.037 143      523      DB      OUT-*     6 - OUTPUT
001.040 165      524      DB      SSTEP-*    7 - SINGLE STEP
001.041 220      525      DB      RMEM-*     8 - CASSETTE LOAD
001.042 332      526      DB      WMEM-*     9 - CASSETTE DUMP
001.043 067      527      DB      NEXT-*     + - NEXT
001.044 104      528      DB      LAST-*     - - LAST
001.045 102      529      DB      ABORT-*     * - ABORT
001.046 060      530      DB      R$W-*     / - DISPLAY/ALTER
001.047 116      531      DB      MEMM-*     $ - MEMORY MODE
001.050 034      532      DB      REGM-*     , - REGISTER MODE
                533
                534 **      PROCESS MEMORY/REGISTER ALTERATIONS.
                535 *
                536 *      THIS CODE IS ENTERED IF
                537 *
                538 *      1) AM IN ALTER MODE, AND
                539 *      2) A KEY FROM 0-7 WAS ENTERED.
                540
001.051 017      541 MTRS  RRC
001.052 173      542      MOV      A,E      (A) = VALUE
001.053 332 067 001 543      JC      MTR6      IS REGISTER
001.056 067      544      STC      INDICATE 1ST DIGIT IS IN (A)
001.057 315 066 003 545      CALL     IOB      INPUT OCTAL BYTE
001.062 043      546      INX      H      DISPLAY NEXT LOCATION

```


PAM/8 - H8 FRONT PANEL MONITOR #01.00.00,
MTR - MAIN EXECUTIVE LOOP.

HEATH X8ASH V1.0 02/18/77
13:23:39 01-APR-77 PAGE 14

		548	**	SAE - STORE ABUSS AND EXIT.		
		549	*			
		550	*	ENTRY	(HL) =	ABUSS VALUE
		551	*	EXIT	TO (RET)	
		552	*	USES	NONE	
		553				
001.063	042 024 040	554	SAE	SHLD	ABUSS	
001.066	311	555		RET		
		556				
		557	*	ALTER REGISTER		
		558				
001.067	365	559	MTR6	PUSH	PSW	SAVE CODE
001.070	315 047 003	560		CALL	LRA	LOCATE REGISTER ADDRESS
001.073	247	561		ANA	A	
001.074	312 322 000	562		JZ	ERROR	NOT ALLOWED TO ALTER STACK POINTER
001.077	043	563		INX	H	
001.100	361	564		POP	PSW	RESTORE VALUE AND CARRY FLAG
001.101	303 062 003	565		JMP	IOA	INPUT OCTAL ADDRESS

PAM/B - H8 FRONT PANEL MONITOR *01.00.00.
MONITOR TASK SUBROUTINES.

HEATH X8ASM V1.1 06/21/77
15:44:14 01-APR-77 PAGE 15

```

569 **      REGM - ENTER REGISTER DISPLAY MODE.
570 *
571 *      ENTRY  (A) = DSPMOD
572 *            (BC) = #DSPMOD
573
001.104 076 002 574 REGM  MVI  A,2      SET DISPLAY REGISTER MODE
040.007 575 .      SET  DSPMOD
001.106 002 576      STAX  B          SET DISPLAY REGISTER MODE
000.000 577      ERRNZ DSPMOD-DSPROT-1
001.107 013 578      DCX   B          (BC) = #DSPROT
001.110 257 579      XRA   A
001.111 002 580      STAX  B          SET ALL PERIODS ON
001.112 315 260 003 581      CALL RCK      READ KEY ENTRY
001.115 075 582      DCR   A          DISPLACE
001.116 376 006 583      CPI   6
001.120 322 322 000 584      JNC   ERROR      NOT 1-6
001.123 007 585      RLC
001.124 022 586      STAX  D          SET NEW REG IND
040.005 587 .      SET   REGI
001.125 311 588      RET

590 **      R$W - TOGGLE DISPLAY/ALTER MODE.
591 *
592 *      ENTRY  (A) = DSPMOD
593 *            (BC) = ADDRESS OF DSPMOD
594
040.007 595 .      SET   DSPMOD
001.126 356 001 596 R$W  XRI   1
001.130 002 597      STAX  B
001.131 311 598      RET

600 **      NEXT - INCREMENT DISPLAY ELEMENT.
601 *
602 *      ENTRY  (HL) = (ABUSS)
603 *            (DE) = ADDRESS OF REGIND
604
001.132 043 605 NEXT  INX   H
001.133 312 063 001 606      JZ    SAE          IF MEMORY, STORE ABUSS AND EXIT
607
608 *      IS REGISTER MODE.
609
040.005 610 .      SET   REGI
001.136 032 611      LDAX  D          (A) = REGI
001.137 306 002 612      ADI   2          INCREMENT REG INNX
001.141 022 613      STAX  D          WRAP TO *SP*
001.142 376 014 614      CPI   12
001.144 330 615      RC     IF NOT TOO LARGE, EXIT
001.145 257 616      XRA   A          OVERFLOW
001.146 022 617      STAX  D
001.147 311 618 ABORT  RET

```

```

620 **      LAST - DECREMENT DISPLAY ELEMENT.
621 *
622 *      ENTRY  (HL) = (ABUSS)
623 *            (DE) = ADDRESS OF REGIND
624
001.150 053      625 LAST  DCX   H
001.151 312 063 001 626 JZ    SAE           IF MEMORY, STORE AND EXIT
627
628 *      IS REGISTER MODE.
629
040.005      630      SET    REGI
001.154 032      631 LST2  LDAX  D           (A) = REGI
001.155 326 002      632 SUI   2
001.157 022      633 STAX  D
001.160 320      634      RNC           IF OK
001.161 076 012      635 MVI   A,10      UNDERFLOW TO *PC*
001.163 022      636 STAX  D
001.164 311      637 RET
638

```

```

640 **      MEMM - ENTER DISPLAY MEMORY MODE.
641 *
642 *      ENTRY  (BC) = ADDRESS OF DSPMOD
643
001.165 257      644 MEMM  XRA    A           (A) = 0
040.007      645      SET    DSPMOD
001.166 002      646 STAX  B           SET DISPLAY MEMORY MODE
000.000      647 ERRNZ  DSPMOD-DSPROT-1
001.167 013      648 DCX   B           (BC) = #DSPROT
001.170 002      649 STAX  B           SET ALL PERIODS ON
001.171 041 025 040 650 LXI   H,ABUSS+1
001.174 303 062 003 651 JMP    IOA           INPUT OCTAL ADDRESS

```

```

653 **      IN - INPUT DATA BYTE.
654 *
655
656 **      OUT - OUTPUT DATA BYTE.
657 *
658 *      ENTRY  (HL) = (ABUSS)
659
001.177 006 333      660 IN    MVI   B,MI.IN
001.201 021      661 DB     MI,LXID      SKIP NEXT INSTRUCTION
001.202 006 323      662 OUT   MVI   B,MI.OUT
001.204 174      663 MOV    A,H           (A) = VALUE
001.205 145      664 MOV    H,L           (H) = PORT
001.206 150      665 MOV    L,B           (L) = IN/OUT INSTRUCTION
001.207 042 002 040 666 SHLD  IOWRK
001.212 315 002 040 667 CALL  IOWRK      PERFORM IO
001.215 154      668 MOV    L,H           (L) = PORT
001.216 147      669 MOV    H,A           (H) = VALUE
001.217 303 063 001 670 JMP    SAE           STORE ABUSS AND EXIT

```

PAM/B - H8 FRONT PANEL MONITOR #01.00.00.

HEATH X8ASM V1.0 02/18/77

GO AND *STEP* FUNCTIONS

13:23:43 01-APR-77 PAGE 18

```

675 **      GO - RETURN TO USER MODE
676 *
677 *      ENTRY  NONE
678
001.222 303 063 000 679 GO      JMP      GO,      ROUTINE IS IN WASTE SPACE
680

681 **      SSTEP - SINGLE STEP INSTRUCTION.
682 *
683 *      ENTRY  NONE
684
001.225      685 SSTEP EQU      *      SINGLE STEP
001.225 363      686      DI      DISABLE INTERRUPTS UNTIL THE RIGHT TIME
001.226 072 011 040 687      LDA      CTLFLG
001.231 356 020      688      XRI      CB.SSI      CLEAR SINGLE STEP INHIBIT
001.233 323 360      689      OUT      OP.CTL      PRIME SINGLE STEP INTERRUPT
001.235 062 011 040 690 SST1   STA      CTLFLG      SET NEW FLAG VALUES
001.240 341      691      POP      H      CLEAN STACK
001.241 303 172 000 692      JMP      INTXIT      RETURN TO USER ROUTINE FOR STEP
693

```

```

694 **      STPRTN - SINGLE STEP RETURN
695
001.244      696 STPRTN EQU      *
001.244 366 020      697      ORI      CB.SSI      DISABLE SINGLE STEP INTERRUPTION
001.246 323 360      698      OUT      OP.CTL      TURN OFF SINGLE STEP ENABLE
040.011      699      SET      CTLFLG
001.250 022      700      STAX      D
001.251 346 040      701      ANI      CB.MTL      SEE IF IN MONITOR MODE
001.253 302 344 000 702      JNZ      MTR
001.256 303 042 040 703      JMP      UIVEC+3      TRANSFER TO USER'S ROUTINE
704

```

```

705 **      RMEM - LOAD MEMORY FROM TAPE.
706 *
707
001.261 041 244 002 708 RMEM   LXI      H,TFABT
001.264 042 031 040 709      SHLD   TFPERR      SETUP ERROR EXIT ADDRESS
710 *      JMP      LOAD

```

```

712 ***      LOAD - LOAD MEMORY FROM TAPE.
713 *
714 *      READ THE NEXT RECORD FROM THE CASSETTE TAPE.
715 *
716 *      USE THE LOAD ADDRESS IN THE TAPE RECORD.
717 *
718 *      ENTRY (HL) = ERROR EXIT ADDRESS
719 *      EXIT  USER P-REG (IN STACK) SET TO ENTRY ADDRESS
720 *           TO CALLER IF ALL OK
721 *           TO ERROR EXIT IF TAPE ERRORS DETECTED.
722 *
723 *
001.267      724 LOAD EQU *
001.267 001.000 376 725 LXI B,1000A-RT.MI*256-256 (BC) = - REQUIRED TYPE AND #
001.272 315 265 002 726 LOAO CALL SRS SCAN FOR RECORD START
001.275 157 727 MOV L,A (HL) = COUNT
001.276 353 728 XCHG (DE) = COUNT, (HL) = TYPE AND #
001.277 015 729 DCR C (C) = - NEXT #
001.300 011 730 DAD B
001.301 174 731 MOV A,H
001.302 305 732 PUSH B SAVE TYPE AND #
001.303 365 733 PUSH PSW SAVE TYPE CODE
001.304 346 177 734 ANI 177H CLEAR END FLAG BIT
001.306 265 735 ORA L
001.307 076 002 736 MVI A,2 SEQUENCE ERROR
001.311 302 205 002 737 JNE TPERR IF NOT RIGHT TYPE OR SEQUENCE
001.314 315 325 002 738 CALL RNP READ ADDR
001.317 104 739 MOV B,H
001.320 117 740 MOV C,A (BC) = P-REG ADDRESS
001.321 076 012 741 MVI A,10
001.323 325 742 PUSH D SAVE (DE)
001.324 315 052 003 743 CALL LRA. LOCATE REG ADDRESS
001.327 321 744 POP D RESTORE (DE)
001.330 161 745 MOV M,C SET P-REG IN MEM
001.331 043 746 INX H
001.332 160 747 MOV M,B
001.333 315 325 002 748 CALL RNP READ ADDRESS
001.336 157 749 MOV L,A (HL) = ADDRESS, (DE) = COUNT
001.337 042 000 040 750 SHLD START
751 *
001.342 315 331 002 752 LOAI CALL RNB READ BYTE
001.345 167 753 MOV M,A
001.346 042 024 040 754 SHLD ABUSS SET ABUSS FOR DISPLAY
001.351 043 755 INX H
001.352 033 756 DCX D
001.353 172 757 MOV A,D
001.354 263 758 ORA E
001.355 302 342 001 759 JNZ LOAI IF MORE TO GO
760 *
001.360 315 172 002 761 CALL CTC CHECK TAPE CHECKSUM
762 *
763 *      READ NEXT BLOCK
764 *
001.363 361 765 POP PSW (A) = FILE TYPE BYTE
001.364 301 766 POP B (BC) = -(LAST TYPE, LAST #)
001.365 007 767 RLC
  
```

PAM/B - H8 FRONT PANEL MONITOR #01.00.00.
LOAD - LOAD MEMORY FROM TAPE

HEATH XBASM V1.1 06/21/77
15:44:21 01-APR-77 PAGE 20

001.366 332 133 002 768
001.371 303 272 001 769

JC TFT
JMP LOAD

ALL DONE - TURN OFF TAPE
READ ANOTHER RECORD

PAM/8 - HS FRONT PANEL MONITOR #01.00.00.
DUMP - DUMP MEMORY TO MAG/PAPER TAPE

HEATH X8ASM V1.0 02/18/77
13:23:47 01-APR-77 PAGE 21

```

772 ***      DUMP - DUMP MEMORY TO MAG TAPE.
773 *
774 *      DUMP SPECIFIED MEMORY RANGE TO MAG TAPE.
775 *
776 *      ENTRY      (START) = START ADDRESS
777 *                (ARUSS) = END ADDRESS
778 *                USER PC = ENTRY POINT ADDRESS
779 *      EXIT      TO CALLER.
780
781
001.374      782 WMEM      EQU      *
001.374 041 244 002 783      LXI      H,IPART
001.377 042 031 040 784      SHLD     IPERRX      SETUP ERROR EXIT
785
002.002 076 001      786 DUMP      MVI      A,UCI,IE
002.004 323 371      787      OUT      OP,TPC      SETUP TAPE CONTROL
002.006 076 026      788      MVI      A,A,SYN
002.010 046 040      789      MVI      H,32      (H) = # OF SYNC CHARACTERS
002.012 315 024 003 790 WME1      CALL     WNB
002.015 045      791      DCR      H
002.016 302 012 002 792      JNZ      WME1      WRITE SYN HEADER
002.021 076 002      793      MVI      A,A,STX
002.023 315 024 003 794      CALL     WNB      WRITE STX
002.026 154      795      MOV      L,H      (HL) = 00
002.027 042 027 040 796      SHLD     CRCSUM      CLEAR CRC 16
002.032 041 001 201 797      LXI      H,RT,MI+80H*256+1      FIRST AND LAST M1 RECORD
002.035 315 017 003 798      CALL     WNF      WRITE HEADER
002.040 052 000 040 799      LHLD     START
002.043 353      800      XCHG      (D,E) = START ADDRESS
002.044 052 024 040 801      LHLD     ABUSS      (H,L) = STOP ADDR
002.047 043      802      INX      H      COMPUTE WITH STOP+1
002.050 175      803      MOV      A,L
002.051 223      804      SUB      E
002.052 157      805      MOV      L,A
002.053 174      806      MOV      A,H
002.054 232      807      SBB      D
002.055 147      808      MOV      H,A      (HL) = COUNT
002.056 315 017 003 809      CALL     WNF      WRITE COUNT
002.061 345      810      PUSH     H
002.062 076 012      811      MVI      A,10
002.064 325      812      PUSH     D      SAVE (DE)
002.065 315 052 003 813      CALL     LRA      LOCATE P-REG ADDRESS
002.070 176      814      MOV      A,M
002.071 043      815      INX      H
002.072 146      816      MOV      H,M
002.073 157      817      MOV      L,A      (HL) = CONTENTS OF PC
002.074 315 017 003 818      CALL     WNF      WRITE HEADER
002.077 341      819      POP      H      (HL) = ADDRESS
002.100 321      820      POP      D      (DE) = COUNT
002.101 315 017 003 821      CALL     WNF
822
002.104 176      823 WME2      MOV      A,M
002.105 315 024 003 824      CALL     WNB      WRITE BYTE
002.110 042 024 040 825      SHLD     ABUSS      SET ADDRESS FOR DISPLAY
002.113 043      826      INX      H
002.114 033      827      DCX      D

```

PAM/8 - H8 FRONT PANEL MONITOR #01.00.00.
DUMP - DUMP MEMORY TO MAG/PAPER TAPE

HEATH XBASM V1.0 02/18/77
13:23:49 01-APR-77 PAGE 22

```

002.115 172      828      MOV      A,D
002.116 263      829      ORA      E
002.117 302 104 002 830      JNZ      WME2          IF MORE TO GO
                        831
                        832 *      WRITE CHECKSUM
                        833
002.122 052 027 040 834      LHLD     CRCSUM
002.125 315 017 003 835      CALL     WNP          WRITE IT
002.130 315 017 003 836      CALL     WNP          FLUSH CHECKSUM
                        837 *      JMP      TFT

                        839 **      TFT - TURN OFF TAPE.
                        840 *
                        841 *      STOP THE TAPE TRANSPORT.
                        842 *
                        843
002.133 257      844 TFT      XRA      A
002.134 323 371  845      OUT      DP,TPC          TURN OFF TAPE

                        847 **      HORN - MAKE NOISE.
                        848 *
                        849 *      ENTRY      (A) = (MILLISECOND COUNT)/2
                        850 *      EXIT      NONE
                        851 *      USES      A,F
                        852
                        853
002.136 076 144  854 ALARM    MVI      A,200/2          200 MS BEEP
002.140 365      855 HORN     PUSH     PSW
002.141 076 200  856      MVI      A,CB,SPK          TURN ON SPEAKER
                        857
002.143 343      858 HRNO     XTHL                      SAVE (HL), (H) = COUNT
002.144 325      859      PUSH     D                  SAVE (DE)
002.145 353      860      XCHG                      (D) = LOOP COUNT
002.146 041 011 040 861      LXI      H,CTLFLG
002.151 256      862      XRA      M
002.152 136      863      MOV      E,M          (E) = OLD CTLFLG VALUE
002.153 167      864      MOV      M,A          TURN ON HORN
002.154 056 033  865      MVI      L,#TICCNT
                        866
002.156 172      867      MOV      A,D          (A) = CYCLE COUNT
002.157 206      868      ADD      M
002.160 276      869 HRN2     CMP      M          WAIT REQUIRED TICCOUNTERS
002.161 302 160 002 870      JNE      HRN2
002.164 056 011  871      MVI      L,#CTLFLG
002.166 163      872      MOV      M,E          TURN HORN OFF
002.167 321      873      POP      D
002.170 341      874      POP      H
002.171 311      875      RET

```



```

880 **      CTC - VERIFY CHECKSUM.
881 *
882 *      ENTRY TAPE JUST BEFORE CRC
883 *      EXIT TO CALLER IF OK
884 *      TO *TPERR* IF BAD
885 *      USES A,F,H,L
886
887
002.172 315 325 002 888 CTC      CALL RNP      READ NEXT PAIR
002.175 052 027 040 889      LHLI      CRCSUM
002.200      174      890      MOV      A,H
002.201      265      891      ORA      L
002.202      310      892      RZ          RETURN IF OK
002.203 076 001      893      MVI      A,1      CHECKSUM ERROR
894 *      JMP      TPERR      (B) = CODE

```

```

896 **      TPERR - PROCESS TAPE ERROR.
897 *
898 *      DISPLAY ERR NUMBER IN LOW BYTE OF ABUSS
899 *
900 *      IF ERROR NUMBER EVEN, DON'T ALLOW #
901 *      IF ERROR NUMBER ODD, ALLOW #
902 *
903 *      ENTRY (A) = NUMBER
904
905
002.205 062 024 040 906 TPERR  STA      ABUSS
002.210      107      907      MOV      B,A      (B) = CODE
002.211 315 133 002 908      CALL     TFT      TURN OFF TAPE
909
910 *      IS #, RETURN (IF PARITY ERROR)
911
002.214      346      912      DB      M1,ANI      FALL THROUGH WITH CARRY CLEAR
002.215      170      913 TER3    MOV      A,B
914
002.216      017      915      RRC
002.217      330      916      RC          RETURN IF OK
917
918 *      BEEP AND FLASH ERROR NUMBER
919
002.220 334 136 002 920 TER1    CC      ALARM      ALARM IF PROPER TIME
002.223 315 252 002 921      CALL     TPXIT      SEE IF #
002.226 333 360      922      IN      IP,FAD
002.230 376 057      923      CPI      00101111B      CHECK FOR #
002.232 312 215 002 924      JE      TER3      IF #
002.235 072 034 040 925      LDA      TICCNT+1
002.240      037      926      RAR
002.241 303 220 002 927      JMP      TER1

```

PAM/8 - H8 FRONT PANEL MONITOR #01.00.00.
TAPE PROCESSING SUBROUTINES

HEATH X8ASM V1.0 02/18/77
13:23:52 01-APR-77 PAGE 24

```

929 **      TPART - ABORT TAPE LOAD OR DUMP.
930 *
931 *      ENTERED WHEN LOADING OR DUMPING, AND THE '*' KEY
932 *      IS STRUCK.
933
934
002,244 257      935 TPART  XRA      A
002,245 323 371  936      OUT      DP,TPC      OFF TAPE
002,247 303 322 000 937      JMP      ERROR

```

```

939 **      TPXIT - CHECK FOR USER FORCED EXIT.
940 *
941 *      TPXIT CHECKS FOR AN '*' KEYPAD ENTRY. IF SO, TAKE
942 *      THE TAPE DRIVER ABNORMAL EXIT.
943 *
944 *      ENTRY  NONE
945 *      EXIT   TO *RET* IF NOT '*'
946 *             (A) = PORT STATUS
947 *             TO (TPERRX) IF '*' DOWN
948 *      USES   A,F
949
950
002,252 333 360  951 TPXIT  IN      IF,PAD
002,254 376 157  952      CPI      01101111B      *
002,256 333 371  953      IN      IF,TPC      READ TAPE STATUS
002,260 300      954      RNE      NOT '*', RETURN WITH STATUS
002,261 052 031 040 955      LHL      TPERRX
002,264 351      956      FCHL      ENTER (TPERRX)

```

```

958 **      SRS - SCAN RECORD START
959 *
960 *      SRS READS BYTES UNTIL IT RECOGNIZES THE START OF A RECORD.
961 *
962 *      THIS REQUIRES
963 *      AT LEAST 10 SYNC CHARACTERS
964 *      1 STX CHARACTER.
965 *
966 *      THE CRC-16 IS THEN INITIALIZED.
967 *
968 *      ENTRY  NONE
969 *      EXIT   TAPE POSITIONED (AND MOVING), CRC SUM = 0
970 *             (DE) = HEADER BYTES
971 *             (HA) = RECORD COUNT
972 *      USES   A,F,D,E,H,L
973
974
002,265      975 SRS     EQU      *
002,265 026 000  976 SRS1   MVI      D,0
002,267 142      977      MOV      H,D
002,270 152      978      MOV      L,D      (HL) = 0

```

```

002.271 315 331 002 979 SRS2 CALL RNB READ NEXT BYTE
002.274 024 980 INR D
002.275 376 024 981 CPI A,SYN
002.277 312 271 002 982 JE SRS2 HAVE SYN
002.302 376 002 983 CPI A,STX
002.304 302 265 002 984 JNE SRS1 NOT STX - START OVER
985
002.307 076 012 986 MVI A,10
002.311 272 987 CMP D SEE IF ENOUGH SYN CHARACTERS
002.312 322 265 002 988 JNC SRS1 NOT ENOUGH
002.315 042 027 040 989 SHLD CRCSUM CLEAR CRC-16
002.320 315 325 002 990 CALL RNP READ LEADER
002.323 124 991 MOV D,H
002.324 137 992 MOV E,A
993 * JMP RNP READ COUNT

995 ** RNP - READ NEXT PAIR.
996 *
997 * RNP READS THE NEXT TWO BYTES FROM THE INPUT DEVICE.
998 *
999 * ENTRY NONE
1000 * EXIT (H,A) = BYTE PAIR
1001 * USES A,F,H
1002
1003
002.325 315 331 002 1004 RNP CALL RNB READ NEXT BYTE
002.330 147 1005 MOV H,A
1006 * JMP RNB READ NEXT BYTE

1008 ** RNB - READ NEXT BYTE
1009 *
1010 * RNB READS THE NEXT SINGLE BYTE FROM THE INPUT DEVICE.
1011 * THE CHECKSUM IS TAKEN FOR THE CHARACTER.
1012 *
1013 * ENTRY NONE
1014 * EXIT (A) = CHARACTER
1015 * USES A,F
1016
1017
002.331 076 064 1018 RNB MVI A,UCI,RO+UCI,ER+UCI,RE TURN ON PEADER FOR NEXT BYTE
002.333 323 371 1019 OUT DP,TPC
002.335 315 252 002 1020 RNB1 CALL TPXIT CHECK FOR * READ STATUS
002.340 346 002 1021 ANI USR,RNR
002.342 312 335 002 1022 JZ RNB1 IF NOT READY
002.345 333 370 1023 IN TP,TPD INPUT DATA
1024 * JMP CRC CHECKSUM

```

PAM/8 - H8 FRONT PANEL MONITOR #01.00.00.
TAPE PROCESSING SUBROUTINES

HEATH X8ASH V1.0 02/18/77
13:23:56 01-APR-77 PAGE 26

```

1026 **      CRC - COMPUTE CRC-16
1027 *
1028 *      CRC COMPUTES A CRC-16 CHECKSUM FROM THE POLYNOMIAL
1029 *
1030 *      (X + 1) * (X15 + X + 1)
1031 *
1032 *      SINCE THE CHECKSUM GENERATED IS A DIVISION REMAINDER,
1033 *      A CHECKSUMED DATA SEQUENCE CAN BE VERIFIED BY RUNNING
1034 *      THE DATA THROUGH CRC, AND THEN RUNNING THE PREVIOUSLY OBTAINED
1035 *      CHECKSUM THROUGH CRC. THE RESULTANT CHECKSUM SHOULD BE 0.
1036 *
1037 *      ENTRY      (CRCSUM) = CURRENT CHECKSUM
1038 *                (A) = BYTE
1039 *      EXIT      (CRCSUM) UPDATED
1040 *                (A) UNCHANGED.
1041 *      USES      F
1042
1043
1044 CRC      PUSH      B          SAVE (BC)
1045          MVI      B,B        (B) = BIT COUNT
1046          PUSH      H
1047          LHLD     CRCSUM
1048 CRC1     RLC
1049          MOV      C,A          (C) = BIT
1050          MOV      A,L
1051          ADD      A
1052          MOV      L,A
1053          MOV      A,H
1054          RAL
1055          MOV      H,A
1056          RAL
1057          XRA      C
1058          RRC
1059          JNC     CRC2          IF NOT TO XOR
1060          MOV      A,H
1061          XRI      200H
1062          MOV      H,A
1063          MOV      A,L
1064          XRI      50H
1065          MOV      L,A
1066 CRC2     MOV      A,C
1067          DCR      B
1068          JNZ     CRC1          IF MORE TO GO
1069          SHLD     CRCSUM
1070          POP      H          RESTORE (HL)
1071          POP      B          RESTORE (BC)
1072          RET              EXIT

```

```

1074 **      WNP - WRITE NEXT PAIR.
1075 *
1076 *      WNP WRITES THE NEXT TWO BYTES TO THE CASSETTE DRIVE.
1077 *
1078 *      ENTRY (H,L) = BYTES
1079 *      EXIT  WRITTEN.
1080 *      USES  A,F
1081
1082
003.017 174 1083 WNP    MOV    A,H
003.020 315 024 003 1084    CALL  WNB
003.023 175 1085    MOV    A,L
1086 *      JMP     WNB          WRITE NEXT BYTE

```

```

1088 **      WNB - WRITE BYTE
1089 *
1090 *      WNB WRITES THE NEXT BYTE TO THE CASSETTE TAPE.
1091 *
1092 *      ENTRY (A) = BYTE
1093 *      EXIT  NONE.
1094 *      USES  F
1095
1096
003.024 365 1097 WNB    PUSH    PSW
003.025 315 252 002 1098 WNB1   CALL    TPXIT      CHECK FOR *, READ STATUS
003.030 346 001 1099    ANI     USR, TXR
003.032 312 025 003 1100    JZ      WNB1          IF MORE TO GO
003.035 076 021 1101    MVI     A, UC1.ER+UC1.1E  ENABLE TRANSMITTER
003.037 323 371 1102    OUT     DP, TPC      TURN ON TAPE
003.041 361 1103    POP     PSW
003.042 323 370 1104    OUT     DP, TPD      OUTPUT DATA
003.044 303 347 002 1105    JMP     CRC          COMPUTE CRC

```

PAM/B - HB FRONT PANEL MONITOR #01.00.00.
SUBROUTINES

HEATH XBASM V1.0 02/18/77
13:23:59 01-APR-77 PAGE 28

```

1109 **      LRA - LOCATE REGISTER ADDRESS.
1110 *
1111 *      ENTRY  NONE.
1112 *      EXIT   (A) = REGISTER INDEX
1113 *            (H,L) = STORAGE ADDRESS
1114 *            (D,E) = (0,A)
1115 *      USES   A,D,E,H,L,F
1116 *
1117 *
1118 *
003.047 072 005 040 1119 LRA    LDA    REGI
003.052 137          1120 LRA:   MOV    E,A
003.053 026 000      1121          MVI    D,0
003.055 052 035 040 1122          LHLH   REGPTR
003.060 031          1123          DAD    D           (DE) = (REGPTR)+(REGI)
003.061 311          1124          RET

```

```

1126 **      IOA - INPUT OCTAL ADDRESS.
1127 *
1128 *      ENTRY  (H,L) = ADDRESS OF RECEPTION DOUBLE BYTE.
1129 *      EXIT   TO *RET* IF ERROR.
1130 *            TO *RET*+1 IF OK, VALUE IN MEMORY.
1131 *      USES   A,D,E,H,L,F
1132 *
1133 *
003.062 315 066 003 1134 IOA    CALL   IOB           INPUT BYTE
003.065 053          1135          DCX    H

```

```

1137 **      IOB - INPUT OCTAL BYTE.
1138 *
1139 *      READ ONE OCTAL BYTE FROM THE KEYSET.
1140 *
1141 *      ENTRY  (H,L) = ADDRESS OF BYTE TO HOLD VALUE
1142 *            PC SET IF FIRST DIGIT IN (A)
1143 *      EXIT   TO *RET* IF ALL OK
1144 *            TO *ERROR* IF ERROR
1145 *      USES   A,D,E,H,L,F
1146 *
1147 *
1148 *
003.066 026 003      1149 IOB    MVI    D,3           (D) = DIGIT COUNT
003.070 324 260 003 1150 IOB:   CNC    RCK           READ CONSOLE KEYSET
1151 *
1152 *      CPI    8
1153 *      JNC    ERROR        IF ILLEGAL DIGIT
1154 *
003.100 137          1155          MOV    E,A           (E) = VALUE
003.101 176          1156          MOV    A,M
003.102 007          1157          RLC           SHIFT 3
003.103 007          1158          RLC

```

```

003.104 007      1159      RLC
003.105 346 370   1160      ANI      370R
003.107 263      1161      ORA      E
003.110 167      1162      MOV      M,A      REPLACE
003.111 025      1163      DCR      D
003.112 302 070 003 1164      JNZ      10B1      IF NOT DONE
003.115 076 017   1165      MVI      A,30/2      BEEP FOR 30 MS
003.117 303 140 002 1166      JMP      HORN

```

```

1168 **      DOD - DECODE FOR OCTAL DISPLAY,
1169 *
1170 *      ENTRY (H,L) = ADDRESS OF LED REFRESH AREA
1171 *      (B) = *OR* PATTERN TO FORCE ON BARS OR PERIODS
1172 *      (A) = OCTAL VALUE
1173 *      EXIT (H,L) = NEX DIGIT ADDRESS
1174 *      USES A,B,C,D,H,L
1175
1176
003.122 325      1177 DOD      PUSH      D
003.123 026 003   1178      MVI      D,DOD/A/256
003.125 016 003   1179      MVI      C,3
003.127 027      1180 DOD1     RAL      LEFT 3 PLACES
003.130 027      1181      RAL
003.131 027      1182      RAL
003.132 365      1183      PUSH      PSW      SAVE FOR NEXT DIGIT
003.133 346 007   1184      ANI      Z
003.135 306 356   1185      ADI      #DOD/A
003.137 137      1186      MOV      E,A      (D) = INDEX
003.140 032      1187      LDAX      D      (A) = PATTERN
003.141 250      1188      XRA      B
003.142 346 177   1189      ANI      177R
003.144 250      1190      XRA      B
003.145 167      1191      MOV      M,A      SET IN MEMORY
003.146 043      1192      INX      H
003.147 170      1193      MOV      A,B
003.150 007      1194      RLC
003.151 107      1195      MOV      B,A
003.152 361      1196      POP      PSW      (A) = VALUE
003.153 015      1197      DCR      C
003.154 302 127 003 1198      JNZ      DOD1      IF MORE TO GO
003.157 321      1199      POP      D
003.160 311      1200      RET      RETURN

```

PAM/B - H8 FRONT PANEL MONITOR #01.00.00.
 UFD - UPDATE FRONT PANEL DISPLAYS.

HEATH XBASH V1.0 02/18/77
 13:24:02 01-APR-77 PAGE 30

```

1203 **      UFD - UPDATE FRONT PANEL DISPLAYS.
1204 *
1205 *
1206 *      UFD IS CALLED BY THE CLOCK INTERRUPT PROCESSOR WHEN IT IS
1207 *      TIME TO UPDATE THE DISPLAY CONTENTS. CURRENTLY, THIS IS DONE
1208 *      EVERY 32 INTERRUPTS, OR ABOUT 32 TIMES A SECOND.
1209 *
1210 *      ENTRY (H,L) = ADDRESS OF REFCNT
1211 *      EXIT  NONE
1212 *      USES  ALL
1213
1214
003.161      1215 UFD      EQU      *
003.161 076 002 1216      MVI      A,00.DDU
003.163 240      1217      ANA      B
003.164 300      1218      RNZ              IF NOT TO HANDLE UPDATE
1219
003.165 056 006 1220      MVI      L,#DISPROT
003.167 176      1221      MOV      A,M
003.170 007      1222      RLC
003.171 167      1223      MOV      M,A      ROTATE PATTERN
003.172 107      1224      MOV      B,A
003.173 043      1225      INX      H
000.000      1226      ERRNZ    DSPMOD-DSPROT-1
003.174 176      1227      MOV      A,M      (A) = DSPMOD
003.175 346 002 1228      ANI      2
003.177 052 024 040 1229      LHLD    ABUSS
003.202 312 227 003 1230      JZ      UFD1      IF MEMORY
1231
1232 *      AM DISPLAYING REGISTERS.
1233
003.205 315 047 003 1234      CALL    LRA      LOCATE REGISTER ADDRESS
003.210 345      1235      PUSH    H
003.211 041 342 003 1236      LXI      H,DSPA
003.214 031      1237      DAD      D      (H,L) = ADDRESS OF REG NAME PATTERNS
003.215 176      1238      MOV      A,M
003.216 043      1239      INX      H
003.217 146      1240      MOV      H,M
003.220 157      1241      MOV      L,A      (H,L) = REG NAME PATTERN
003.221 343      1242      XTHL
003.222 264      1243      ORA      H      CLEAR Z'
003.223 176      1244      MOV      A,M
003.224 043      1245      INX      H
003.225 146      1246      MOV      H,M
003.226 157      1247      MOV      L,A      (HL) = ADDRESS OF REGISTER PAIR CONTENTS
1248
1249 *      SETUP DISPLAY
1250
003.227 365      1251 UFD1    PUSH    PSW
003.230 353      1252      XCHG
003.231 041 013 040 1253      LXI      H,ALED5
003.234 172      1254      MOV      A,D
003.235 315 122 003 1255      CALL    DOD      FORMAT ABANK HIGH HALF
003.240 173      1256      MOV      A,E
003.241 315 122 003 1257      CALL    DOD      FORMAT ABANK LOW HALF
003.244 361      1258      POP     PSW

```


PAM/8 - H8 FRONT PANEL MONITOR #01.00.00.
UPD - UPDATE FRONT PANEL DISPLAYS.

HEATH X8ASM V1.0 02/18/77
13:24:04 01-APR-77 PAGE 31

003,245	032	1259	LDAX	D	
003,246	312 122 003	1260	JZ	D0D	IF MEMORY, DECODE BYTE VALUE
		1261			
		1262 *			IS REGISTER. SET REGISTER NAME.
		1263			
003,251	066 377	1264	MVI	M,377H	CLEAR DIGIT
003,253	341	1265	POP	H	
003,254	042 022 040	1266	SHLD	DLEDS+1	
003,257	311	1267	RET		

PAM/B - HB FRONT PANEL MONITOR #01.00.00.

HEATH X8ASH V1.1 06/21/77

RCK - READ CONSOLE KEYPAD.

15:44:39 01-APR-77 PAGE 32

```

1271 **      RCK - READ CONSOLE KEYPAD.
1272 *
1273 *      RCK IS CALLED TO READ A KEYSTROKE FROM THE CONSOLE KEYPAD.
1274 *      WHENEVER A KEY IS ACCEPTED.
1275 *      RCK PERFORMS DEBOUNCING, AND AUTO-REPEAT. A *BIP* IS SOUNDED
1276 *      WHEN A VALUE IS ACCEPTED.
1277 *
1278 *      KEY PAD VALUES:
1279 *
1280 *      1111 1110 - 0
1281 *      1111 1100 - 1
1282 *      1111 1010 - 2
1283 *      1111 1000 - 3
1284 *      1111 0110 - 4
1285 *      1111 0100 - 5
1286 *      1111 0010 - 6
1287 *      1111 0000 - 7
1288 *      1110 1111 - 8
1289 *      1100 1111 - 9
1290 *      1010 1111 - +
1291 *      1000 1111 - -
1292 *      0110 1111 - *
1293 *      0100 1111 - /
1294 *      0010 1111 - #
1295 *      0000 1111 - .
1296 *
1297 *
1298 *      ENTRY  NONE
1299 *      EXIT   TO CALLER WHEN A KEY IS HIT
1300 *      (A) = 0 - '0'
1301 *      1 - '1'
1302 *      2 - '2'
1303 *      3 - '3'
1304 *      4 - '4'
1305 *      5 - '5'
1306 *      6 - '6'
1307 *      7 - '7'
1308 *      8 - '8'
1309 *      9 - '9'
1310 *      10 - '+'
1311 *      11 - '-'
1312 *      12 - '*'
1313 *      13 - '/'
1314 *      14 - '#'
1315 *      15 - '.'
1316 *      USES  A,F
1317
1318
003.260      1319 RCK EQU *
003.260 345 1320 PUSH H
003.261 305 1321 PUSH B
003.262 014 024 1322 MVI C,400/20 WAIT 400 MS
003.264 041 026 040 1323 LXI H,RCKA
1324
003.267 333 360 1325 RCK1 IN IF,PAD INPUT PAD VALUE
003.271 107 1326 MOV B,A (B) = VALUE

```

PAM/S - H8 FRONT PANEL MONITOR #01.00.00.
RCK - READ CONSOLE KEYPAD.

HEATH X8ASM V1.1 06/21/77
15:44:41 01-APR-77 PAGE 33

003.272	076 012	1327	MVI	A,20/2	
003.274	315 053 000	1328	CALL	DLY	WAIT 20 MS
003.277	170	1329	MOV	A,B	
003.300	276	1330	CMF	M	
003.301	302 310 003	1331	JNE	RCK2	HAVE A CHANGE
003.304	015	1332	DCR	C	
003.305	302 267 003	1333	JNZ	RCK1	WAIT N CYCLES
		1334			
		1335	*	HAVE KEY VALUE	
		1336			
003.310	167	1337	RCK2	MOV	M,A
003.311	356 376	1338	XRI	376Q	UPDATE RCKA
003.313	017	1339	RRC		INVERT ALL BUT GROUP 0 FLAG
003.314	322 326 003	1340	JNC	RCK3	HIT BANK 0
003.317	017	1341	RRC		
003.320	017	1342	RRC		
003.321	017	1343	RRC		
003.322	017	1344	RRC		
003.323	322 267 003	1345	JNC	RCK1	NO HIT AT ALL
003.326	107	1346	RCK3	MOV	(B) = CODE
003.327	076 002	1347	MVI	A,4/2	
003.331	315 140 002	1348	CALL	HORN	MAKE BIP
003.334	170	1349	MOV	A,B	
003.335	346 017	1350	ANI	17Q	
003.337	301	1351	POP	B	
003.340	341	1352	POP	H	
003.341	311	1353	RET		RETURN

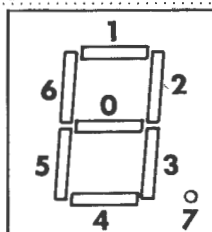


PAM/8 - HB FRONT PANEL MONITOR #01.00.00.
SEGMENT PATTERNS AND CONSTANTS.

HEATH X8ASM V1.1 06/21/77
15:44:42 01-APR-77 PAGE 34

1357 ** DISPLAY SEGMENT CODING:

1358 *
1359 * BYTE = 76 543 210
1360 *
1361 *
1362 *
1363 *
1364 *
1365 *
1366 *



1370 ** REGISTER INDEX TO 7-SEGMENT PATTERN

Address	Register	Pattern	Register	Pattern
003.342	DS	0	DS	0
003.342	244	230	1373	DW 1001100010100100B SF
003.344	220	234	1374	DW 1001110010010000B AF
003.346	206	215	1375	DW 10001101100000110B BC
003.350	302	214	1376	DW 10001100110000010B DE
003.352	222	217	1377	DW 1000111110010010B HL
003.354	230	316	1378	DW 1100111010011000B PC

1380 ** OCTAL TO 7-SEGMENT PATTERN

Address	Register	Pattern	Register	Pattern
003.356	DS	0	1381	DODA DS 0
003.356	001	00000001B	1382	DB 00000001B 0
003.357	163	01110011B	1383	DB 01110011B 1
003.360	110	01001000B	1384	DB 01001000B 2
003.361	140	01100000B	1385	DB 01100000B 3
003.362	062	00110010B	1386	DB 00110010B 4
003.363	044	00100100B	1387	DB 00100100B 5
003.364	004	00000100B	1388	DB 00000100B 6
003.365	161	01110001B	1389	DB 01110001B 7
003.366	000	00000000B	1390	DB 00000000B 8
003.367	040	00100000B	1391	DB 00100000B 9

no true fit instruction

1394 ** I/O ROUTINES TO BE COPIED INTO AND USED IN RAM.

1395 *
1396 * MUST CONTINUE TO 3777A FOR PROPER COPY.
1397 * THE TABLE MUST ALSO BE BACKWARDS TO THE FINAL RAM

Address	Register	Value	Register	Value
003.371	ORG	4000A-7	1398	
003.371	1401	PERSDM EQU *	1399	
003.371	001	1	1400	
003.372	000	0	1401	PERSDM EQU *
003.373	000	0	1402	DB 1
			1403	DB 0
			1404	DB 0

PAM/8 - H8 FRONT PANEL MONITOR #01.00.00.
CONSTANTS AND TABLES.

HEATH X8ASM V1.1 06/21/77
15:44:44 01-APR-77 PAGE 35

003.374	000	1405	DB	0	DSPMOD
003.375	000	1406	DB	0	DSPROT
003.376	012	1407	DB	10	REG1
003.377	311	1408	DB	MI.RET	
		1409			
000.000		1410	ERRNZ	*-4000A	



PAM/8 - H8 FRONT PANEL MONITOR #01.00.00.
RAM CELLS

HEATH X8ASM V1.1 06/21/77
15:44:44 01-APR-77 PAGE 36

```

1413
1414 **      THE FOLLOWING ARE CONTROL CELLS AND FLAGS USED BY THE KEYPAD
1415 *      MONITOR.
1416
040.000      1417      ORG      40000A      8192
040.000      1418      START   DS      2      DUMP STARTING ADDRESS
040.002      1419      IOWRK   DS      2      IN OR OUT INSTRUCTION
040.004      1420      PRSRAM  EQU      *      FOLLOWING CELLS INITIALIZED FROM ROM
040.004      1421      DS      1      RET
1422
040.005      1423      REGI     DS      1      INDEX OF REGISTER UNDER DISPLAY
040.006      1424      DSPROT   DS      1      PERIOD FLAG BYTE
040.007      1425      DSPMOD   DS      1      DISPLAY MODE
1426
040.010      1427      MFLAG    DS      1      USER FLAG OPTIONS
1428      *      SEE *UO.XXX* BITS DESCRIBED AT FRONT
1429
040.011      1430      CTLFLG   DS      1      FRONT PANEL CONTROL BITS
040.012      1431      REFINI   DS      1      REFRESH INDEX (0 TO 7)
000.007      1432      PRSL     EQU      *-PRSRAM  END OF AREA INITIALIZED FROM ROM
1433
040.013      1434      FFLEDS   EQU      *      FRONT PANEL LED PATTERNS
040.013      1435      ALEDS    DS      1      ADDR 0
040.014      1436      DS      1      ADDR 1
040.015      1437      DS      1      ADDR 2
1438
040.016      1439      DS      1      ADDR 3
040.017      1440      DS      1      ADDR 4
040.020      1441      DS      1      ADDR 5
1442
040.021      1443      DLEDS    DS      1      DATA 0
040.022      1444      DS      1      DATA 1
040.023      1445      DS      1      DATA 2
1446
040.024      1447      ABUSS     DS      2      ADDRESS BUS
040.026      1448      RCKA      DS      1      RCK SAVE AREA
040.027      1449      CRCSUM    DS      2      CRC-16 CHECKSUM
040.031      1450      TPERRX    DS      2      TAPE ERROR EXIT ADDRESS
040.033      1451      TICCNT    DS      2      CLOCK TIC COUNTER
1452
040.035      1453      REGPTR    DS      2      REGISPTR CONTENTS POINTER
1454
040.037      1455      UIVEC     DS      0      USER INTERRUPT VECTORS
040.037      1456      DS      3      JUMP TO CLOCK PROCESSOR
040.042      1457      DS      3      JUMP TO SINGLE STEP PROCESSOR
040.045      1458      DS      3      JUMP TO I/O 3
040.050      1459      DS      3      JUMP TO I/O 4
040.053      1460      DS      3      JUMP TO I/O 5
040.056      1461      DS      3      JUMP TO I/O 6
040.061      1462      DS      3      JUMP TO I/O 7
1463
040.064      1464      END
ASSEMBLY COMPLETE
1464 STATEMENTS
0 ERRORS DETECTED
22310 BYTES FREE

```

CROSS REFERENCE TABLE.

XREF V1.0 PAGE 37

	040011	217S	344S	494S	513S	574S	586S	594S	609S	630S	645S	699S
..MFLAG	040010	344	381	386	418	459	465	1427L				
A.STX	000002	93E	793	983								
A.SYN	000026	92E	788	981								
ABORT	001147	529	617L									
ABUSS	040024	490	554	650	754	801	825	906	1229	1447L		
ALARM	002136	469	854L	920								
ALEDS	040013	1253	1435L									
BLKSI2	002000	118E										
CB.CLI	000100	99L	136	256	464							
CB.MTL	000040	78E	331	415	464	701						
CB.SPK	000200	100E	256	464	856							
CB.SSI	000020	97E	256	331	464	688	697					
CLK2	000234	393	395L									
CLK3	000237	389	398E									
CLK4	000313	422	438E									
CLOCK	000201	202	203	369L								
CRC	002347	1044L	1105									
CRC1	002356	1048L	1068									
CRC2	003004	1059	1066L									
CRCSUM	040027	796	834	889	989	1047	1069	1449L				
CTC	002172	761	888L									
CTLFLG	040011	217	328	386	391	413	418	465	687	690	699	861
		1430L										871
CUI1	000165	345L	441									
DLEDS	040021	1266	1443L									
DLY	000053	248L	1328									
DM.MF	000000	104E										
DM.MW	000001	105E										
DM.RR	000002	106E										
DM.RW	000003	107E										
DOD	003122	1177L	1255	1257	1260							
DOD1	003127	1180L	1198									
DODA	003356	1178	1185	1382L								
DSPA	003342	1236	1372L									
DSPMOD	040007	481	494	513	574	576	594	645	647	1226	1425L	
DSPROT	040006	485	576	647	1220	1226	1424L					
DUMP	002002	786L										
ERROR	000322	298	434	458E	503	562	583	937	1153			
FFLEDS	040013	1434E										
GO	001222	521	679L									
GO.	000063	256L	679									
HORN	002140	855L	1166	1348								
HRN0	002143	250	858L									
HRN2	002160	869L	870									
IN	001177	522	660L									
INIT	000073	192	193	275L	279							
INIT0	000000	190L										
INIT1	000107	288L	293									
INIT2	000117	295L										
INT1	000010	197E										
INT2	000020	212E										
INT3	000030	229L										
INT4	000040	234L										
INT5	000050	239L										
INT6	000060	253L										
INT7	000070	260L										
INTXIT	000172	352L	416	692								

CROSS REFERENCE TABLE.

XREF V1.0 PAGE 38

IOA	003062	565	651	1134L					
IOB	003066	545	1134	1149L					
IOB1	003070	1150L	1164						
IOWRK	040002	666	667	1419L					
IP.FAD	000360	81E	439	922	951	1325			
IP.TPC	000371	85E	953						
IP.TPD	000370	87E	1023						
LAST	001150	528	625L						
LOAO	001272	726L	769						
LOA1	001342	752L	759						
LOAD	001267	724E							
LRA	003047	560	1119L	1234					
LRA.	003052	427	743	813	1120L				
LST2	001154	631L							
MEMM	001165	531	644L						
MI.ANI	000346	128E	912						
MI.HLT	000166	123E	433						
MI.IN	000333	125E	660						
MI.LDA	000072	127E							
MI.LXID	000021	129E	661						
MI.OUT	000323	126E	662						
MI.RET	000311	124E	1408						
MTR	000344	476E	702						
MTR1	000345	479	479L						
MTR4	001005	492	502L						
MTR5	001051	497	541L						
MTR6	001067	543	559L						
MTR8	001035	506	520E						
NEXT	001132	527	604L						
OP.CYL	000360	82E	689	698					
OP.DIG	000360	83E	400						
OP.SEG	000361	84E	402						
OP.TPC	000371	86E	304	787	845	936	1019	1102	
OP.TPD	000370	88E	1104						
OUT	001202	523	662L						
PRSL	000007	191	1432E						
PRSRAM	040004	191	1420E	1432					
PRSR0M	003371	190	1401E						
R#W	001126	530	595L						
RCK	003260	489	580	1150	1319E				
RCK1	003267	1325L	1333	1345					
RCK2	003310	1331	1337L						
RCK3	003326	1340	1346L						
RCKA	040026	1323	1448L						
REFIND	040012	391	1431L						
REGI	040005	512	586	609	630	1119	1423L		
REGH	001104	532	573L						
REGPTR	040035	335	467	1122	1453L				
RMEM	001261	525	708L						
RNB	002331	752	979	1004	1018L				
RNB1	002335	1020L	1022						
RNP	002325	738	748	888	990	1004L			
RT.BF	000002	113E							
RT.CT	000003	114E							
RT.MI	000001	112E	725	797					
SAE	001063	554L	605	626	670				
SAVALL	000132	200	215	319L					
SINCR	004000	281E	283	284					



CROSS REFERENCE TABLE.

XREF V1.0 PAGE 39

SRS	002265	726	975E						
SRS1	002265	976L	984	988					
SRS2	002271	979L	982						
SST1	001235	257	690L						
SSTEP	001225	524	685E						
START	040000	284	750	799	1418L				
STPRTN	001244	218	696E						
TER1	002220	920L	927						
TER3	002215	913L	924						
TFT	002133	766	844L	908					
TICCNT	040033	369	371	406	865	925	1451L		
TPABT	002244	708	783	935L					
TPERR	002205	737	906L						
TPERRX	040031	709	784	955	1450L				
TPXIT	002252	921	951L	1020	1098				
UCI.ER	000020	165E	1018	1101					
UCI.IE	000002	167E							
UCI.IR	000100	163E							
UCI.RE	000004	166E	1018						
UCI.RD	000040	164E	1018						
UCI.TE	000001	168E	786	1101					
UFD	003161	409	1215E						
UFD1	003227	1230	1251L						
UIVEC	040037	229	234	239	253	260	348	703	1455L
UMI.16X	000002	158E	303						
UMI.1R	000100	148E	303						
UMI.1X	000001	157E							
UMI.2B	000300	150E							
UMI.64X	000003	159E							
UMI.HB	000200	149E							
UMI.L5	000000	153L							
UMI.L6	000004	154E							
UMI.L7	000010	155E							
UMI.L8	000014	156E	303						
UMI.PA	000020	152E							
UMI.PE	000040	151E							
UD.CLK	000001	138E	346						
UD.DDU	000002	137E	461	1216					
UD.HLT	000200	135E	420						
UD.NFR	000100	136E	384	461					
USR.FE	000040	172E							
USR.DE	000020	173E							
USR.PE	000010	174E							
USR.RXR	000002	176E	1021						
USR.TXE	000004	175E							
USR.TXR	000001	177E	1099						
WME1	002012	790L	792						
WME2	002104	823L	830						
WMEM	001374	526	782E						
WNB	003024	790	794	824	1084	1097L			
WNB1	003025	1098L	1100						
WNP	003017	798	809	818	821	835	836	1083L	

25434 BYTES FREE

APPENDIX B

DEMO: PAM8

This program shows the advanced features of PAM8 and, as such, should not be evaluated as either an efficient or useful routine. The program uses the H8 clock, keyboard, display and interrupt capabilities to create an accurate interval timer that lets you enter an integer value from zero through nine seconds. When the program has counted down to zero, an audio alert is sounded, ending the program and returning control to PAM8.

Use the H8 keypad to enter the machine code, set the program counter, and execute the program. While the program is being executed, the front panel display will be turned off and the computer will wait for you to enter a digit from the keypad. A single digit corresponding to the integer you selected is displayed and decremented until control is returned to PAM8.

The timer is typical of a program you might create. An interval timer, a clock, or even a game requires that you communicate with the H8. The keypad lets you communicate with the CPU, and the CPU uses the LED display to communicate with you. The computer understands the selected time interval when you press a decimal key on the front panel. The job status, or decremented time interval, is relayed to you by the front panel displays. This interaction between you and the machine is characteristic of most software applications.

The program uses the PAM8 firmware. Although it appears simple enough, you must study both the program and the PAM8 listing ("Appendix A") in order to understand what happens when the program is operating. We suggest that you take a course in assembly language programming, such as the Heath EC1108, if you have difficulty understanding the program.

The program source listing was prepared on an H8 computer system using the text editor (TED-8) and the assembler (HASL-8). NOTE: Your programs can be handwritten and assembled if you have only an H8.

The Sample Program

This program initially blanks the LED display and waits for you to enter an integer value. The computer verifies that the value you selected is permissible and then increments and stores the integer. The value was incremented because the display routine always decrements the count by one when it is called.

The most subtle part of this program is the interrupt service routine.* The H8 requires that you initialize the interrupt service routine by loading an instruction and address into the user interrupt vector (UIVEC) before executing the interrupt. After UIVEC is initialized, the program will jump to the service routine after the next interrupt signal is generated.

The main body of the program is a “do-nothing” loop that holds the program in a wait status until the interval timer has reached zero. You could replace the loop with another program which would execute simultaneously with the clock counter. When the countdown is complete, the program returns the H8 computer to its original status before halting.

*NOTE: Basically, an interrupt is a CPU response to a control signal. This signal directs the software to automatically save the current CPU status and transfers program control to a specified routine, called an interrupt handler. When the interrupt handler completes the routine, program control returns to its original status and normal program execution continues.

HEATH ASM #104.01.00.
PAGE 1

*** *****

*
* DEMO: PAMB

*
* SYSTEM DEFINITIONS

*

040.100		ORG	40100A	
000.322	ERROR	EQU	322A	RESET PAMB
002.140	HORN	EQU	2140A	MAKE NOISE
003.260	RCK	EQU	3260A	READ CONSOLE KEYPAD
003.356	DODA	EQU	3356A	OCTAL TO 7-SEGMENT PATTERN
040.010	.MFLAG	EQU	40010A	USER FLAG OPTIONS
040.013	FPLEDS	EQU	40013A	FRONT PANEL L.E.D. PATTERNS
040.037	UIVEC	EQU	40037A	USER INTERRUPT VECTOR
000.001	UO.CLK	EQU	1A	ALLOW CLOCK INTERRUPT PROCESSING
000.002	UO.DDU	EQU	2A	DISABLE DISPLAY UPDATE
000.303	MI.JMP	EQU	303A	MACHINE INSTRUCTION (8080) JUMP
000.377	LEDOFF	EQU	377A	BLANK L.E.D. DISPLAY

*** *****

*

* DISABLE UPDATING OF L.E.D. DISPLAY
* AND TURN OFF L.E.D.'S

*

040.100	076 002	PAMB	MVI	A,UO.DDU	DISABLE NORMAL UPDATING
040.102	062 010 040		STA	.MFLAG	DONE
040.105	041 013 040		LXI	H,FPLEDS	L.E.D. DISPLAY ADDRESS
040.110	006 011		MVI	B,9	COUNT L.E.D.'S
040.112	076 377		MVI	A,LEDOFF	TURN OFF L.E.D.
040.114	167	BLANK	MOV	M,A	O.K. - GO
040.115	043		INX	H	NEXT L.E.D. ADDRESS
040.116	005		DCR	B	ALL DONE - ??
040.117	302 114 040		JNZ	BLANK	NO - DO AGAIN!

*** *****

*

* READ A DECIMAL INTEGER FROM H8 FRONT PANEL
* IF NOT DECIMAL -- RETURN TO PAM-B.
* INCREMENT THE INTEGER (A PROGRAM REQUIREMENT)
* STORE THE DIGIT.

*

040.122	315 260 003		CALL	RCK	READ CONSOLE KEYPAD
040.125	376 012		CPI	10D	TEST IF ZERO THRU NINE
040.127	322 322 000		JNC	ERROR	ABORT TO PAM-B
040.132	074		INR	A	<A>=<A>+1
040.133	062 254 040		STA	DIGIT	STORE INTEGER

*** *****

*

* INITIALIZE CLOCK COUNTER.
* PROGRAM REQUIRES ONE INTERRUPT BEFORE DISPLAY

*

040.136	041 001 000		LXI	H,1	H=0 & L=1
040.141	042 252 040		SHLD	TICK	INITIALIZE COUNT

*** *****

*

* INITIALIZE SERVICE INTERRUPT ROUTINE
* LOAD THE USER INTERRUPT VECTOR (UIVEC) WITH A
* JUMP INSTRUCTION AND THE ADDRESS OF THE SERVICE
* ROUTINE... ENABLE USER CLOCK INTERRUPT!

*

```

*
040.144 076 303 MVI A,M1.JMP SET-UP JUMP INSTRUCTION
040.146 062 037 040 STA UIVEC STORE "JMP" INSTRUCTION
040.151 041 207 040 LXI H,INTRP USER INTERRUPT ADDRESS
040.154 042 040 040 SHLD UIVEC+1 POSITIONED
040.157 076 003 MVI A,U0,DDU+U0,CLK
040.161 062 010 040 STA MFLAG DISABLE UPDATE & ENABLE CLOCK INT.
*** *****
*
* WAIT FOR CLOCK TO REACH ZERO
*
040.164 072 254 040 LOOP LDA DIGIT DO NOTHING LOOP.
040.167 376 000 CFI 0 WAIT FOR END
040.171 302 164 040 JNZ LOOP OF COUNT DOWN.
*** *****
*
* RETURN TO NORMAL INTERRUPT STATUS & HALT.
*
* DISABLE INTERRUPT & TURN ON SPEAKER
*
040.174 076 002 MVI A,U0,DDU
040.176 062 010 040 STA MFLAG DISABLE UPDATE & CLOCK INTERRUPT
040.201 076 372 MVI A,500/2 250 MS BEEP
040.203 315 140 002 CALL HORN
040.206 166 HLT
*** *****
*
* INTERRUPT ROUTINE
*
* CLOCK AND DISPLAY INTERRUPT
*
040.207 052 252 040 INTR LHLD TICK GET COUNT (BETWEEN 0 & 500)
040.212 053 DCX H TICK=TICK-1
040.213 042 252 040 SHLD TICK STORE COUNT
040.216 175 MOV A,L TEST FOR ZERO
040.217 264 RRA H COMPARE WITH 'H'
040.220 300 RNE EXIT IF .NE. 0
*** *****
*
* UPDATE L.E.D. DISPLAY FOR 'NEW' DIGIT.
*
*
040.221 072 254 040 LDA DIGIT GET INTEGER
040.224 075 DCR A DIGIT=DIGIT-1
040.225 062 254 040 STA DIGIT SAVE INTEGER
040.230 041 356 003 LXI H,D0DA DECODE DISPLAY ADDRESS
040.233 205 ADD L POSITION DISPLAY
040.234 157 MOV L,A ALL SET -- GO
040.235 176 MOV A,M DISPLAY SET
040.236 366 200 ORI 2000 MASK - TURN OFF D.P.
040.240 062 017 040 STA FPLEDS+4 TURN-ON-THE-LIGHTS
040.243 041 364 001 LXI H,500 RESTORE COUNT
040.246 042 252 040 SHLD TICK WITH 500
040.251 311 RET
*
* STORAGE AREA & END ASSEMBLY
*
040.252 TICK DS 2
040.254 001 DIGIT DB 1
040.255 000 END PAMB

```

.....00130 STATEMENTS ASSEMBLED.....
12275 BYTES FREE
.....NO ERRORS DETECTED.....



INDEX

- Addressing Port Pairs, 1-21
- Advanced Control, 1-22
- Alter Key, 1-9, 1-12
- Altering a Memory Location, 1-12 ff
- Altering a Selected Register, 1-15
- Audio Alert, 1-9

- Binary to Octal Conversion, 1-7
- Breakpoint Interrupts, 1-24
- Breakpointing, 1-16

- Cancel, 1-9, 1-19
- Checksum Errors, 1-20
- Clock, 1-35, 1-6
- Clock Interrupts, 1-5, 1-24
- Copying a Tape, 1-20

- Decrements Memory, 1-9
- Disable Interrupt, 1-5
- Displays, 1-7, 1-23
- DODA, 1-66
- Dump Routines, 1-17 ff
- Dumping, 1-18, 1-46

- Execution Control, 1-16 ff
- Entry Point, 1-19
- Ending Address, 1-20
- ERROR, 1-66

- FPLEDS, 1-23 (1-60, 1-66)

- GO Key, 1-16

- Halt Instruction, 1-16
- HORN, 1-66

- I/O, 1-21
- I/O Interrupts, 1-24
- IP.PAD, 1-22
- Increment Memory, 1-9
- Initialization, 1-5
- Inputting, 1-21
- Interrupt Vectors, 1-31
- Interrupts, 1-24, 1-26
- Interrupting a Program, 1-17

- Keypad, 1-9, 1-22

- Load Routines, 1-17 ff
- Loading, 1-18, 1-44

- MEM Key, 1-9
- .MFLAG (1-23, 1-66)
- Manual DI, 1-23
- Manual Updating, 1-23
- Master Clear, 1-5
- Monitor Mode, 1-6

- Offset Octal, 1-8
- Outputting, 1-21

- Port I/O, 1-21
- PAM/8 RAM Cells, 1-61
- Power-Up, 1-5

- RAM High Limit, 1-5
- RCK, 1-22, 1-57, 1-66
- REG Key, 1-9
- RST, 1-5 ff, 1-9
- RTM, 1-5 ff, 1-17, 1-9
- Record Errors, 1-20
- Refreshing, 1-23
- Repeats, 1-9

Single Instruction, 1-17
Single Instruction Interrupts, 1-24
Specifying a Memory Address, 1-10 ff
Specifying a Register for Display, 1-14
Stack Space, 1-5
Stepping Through Memory, 1-13
Stepping Through the Registers, 1-15

TICCNT, 1-22

Tape Errors, 1-20, 1-48
Tick Counter, 1-22

UIVEC, (1-60)
USART Bit Definitions, 1-30
User Mode, 1-6
User Option Bits, 1-23, 1-29
UO.CLK, (1-66)
UO.DDU, (1-66)

Section 2

CONSOLE DEBUGGER

BUG-8



TABLE OF CONTENTS

INTRODUCTION	2-3
MEMORY COMMANDS	
The Format Control	2-4
Range	2-6
Displaying Memory Contents	2-7
Altering Memory — Decimal or Octal	2-8
Altering Memory ASCII Format	2-9
REGISTER COMMANDS	
Displaying All Registers	2-10
Displaying Individual Registers	2-10
Altering Register Contents	2-11
EXECUTION CONTROL	
Single Stepping	2-12
Breakpointing	2-13
TAPE HANDLING	
Tape Errors	2-18
TERMINAL CONTROL CHARACTERS	
Input Control Characters	2-19
Output Control Characters	2-19
THE DISCARD FLAG, CNTRL-O AND CNTRL-P	
2-20	
COMMAND COMPLETION	
2-21	
APPENDIX A	
Loading From the Software Distribution Tape	2-22
Loading From a Configured Tape	2-23
APPENDIX B	
Memory Commands	2-24
Range	2-24
Register Commands	2-25
Execution Control	2-26
Tape Handling	2-26
INDEX	
2-27	

INTRODUCTION

The Heath Console Debugger, BUG-8, is designed to allow you to enter and debug machine language programs from a console terminal. BUG-8 occupies the lowest 3K of RAM, starting at 040 100. A user program can be loaded into any free RAM (random access memory) locations, and can be manipulated via BUG-8.

BUG-8 contains facilities to perform the following nine major functions:

- Display the contents of a selected memory location.
- Alter the contents of a selected memory location.
- Display the contents of any 8080 register.
- Alter the contents of any 8080 register.
- Execute the user program a single instruction at a time.
- Execute the program.
- Insert breakpoints and execute the user program.
- Load user programs from tape storage.
- Dump user program to tape storage.

A number of features were designed into BUG-8 for your convenience. Memory locations and memory and register contents may be displayed as bytes or as words, in octal, decimal, or ASCII format. With these features, you can select the most familiar or desirable format. BUG-8 also contains a single instruction facility that permits you to execute your program a single instruction at a time. And for more advanced program analysis, a breakpointing feature is included that permits you to execute several instructions in a program and then return control to BUG-8 for analysis and/or modification.

The standard Heath console driver is incorporated in BUG-8; therefore, it responds to all the normal console input and output control characters. For this reason BUG-8 includes error detection and command completion as outlined on Page 0-33 of the "Introduction" to this Software Reference Manual.

Since BUG-8 is primarily used with assembly and machine language programs, most numeric values are OCTAL. See the individual command description for more details.



MEMORY COMMANDS

The memory commands permit you to display and alter the contents of indicated memory locations. The format for memory display commands is:

< FORMAT CONTROL > < range > < blank >

The form for the alter memory command is:

< FORMAT CONTROL > < range > = < value list >

Format control specifies that memory display/alteration is in word or byte format, and whether octal, decimal or ASCII notation is to be used. The range specifies the memory address or addresses to be displayed or altered, and the command is executed by the typing of a blank using the space bar on the console terminal.

The Format Control

The format control consists of two characters which specify the form of the values that are to be displayed and entered. The format control field may take on a number of different forms. They are:

<u>FORMAT CONTROL</u>	<u>DESCRIPTION</u>
< null > < null >	Display/alter octal integers, byte format.
F < null >	Display/alter as octal integers, word format.
< null > A	Display/alter as ASCII characters, byte format.
FA	Display/alter as ASCII characters, word format.
< null > D	Display/alter as decimal integers, byte format.
FD	Display/alter as decimal integers, word format.

WORD FORMAT (F)

If an F is specified as the first character of the format control field, it indicates that the values are to be displayed/alteres as “full words.” This is to say that memory locations are taken as two byte pairs. The second byte is considered to be the high order (most significant) byte and is displayed first. The first byte is considered to be the low order (least significant) byte and is displayed last.

BYTE FORMAT (NULL)

If an F is not specified, the first character is null, indicating that the values are to be displayed/alteres as single bytes. You can create a NULL by not typing any character for the format control portion of the memory command.

OCTAL FORMAT (NULL)

If no option (a NULL) is specified as the second character of the format control field, the values to be displayed/alteres are taken to be octal integers. The NULL was chosen to specify both byte format and octal notation, as byte octal is the most commonly used format. A blank separates each octal integer, or octal integer pair if the F is specified.

DECIMAL FORMAT (D)

If a D is specified as the second character of the format control field, the values to be displayed/alteres are taken to be decimal integers. A blank separates each decimal integer, or decimal integer pair if the F is specified.

ASCII FORMAT (A)

If an A is specified as the second character of the format control field, the values to be displayed/alteres are converted from/to eight bit representations of ASCII characters. A blank separates each character, or character pair if the F is specified.



Range

The range field consists of a beginning address and an ending address. You can specify addresses by using the appropriate offset octal integers; or you can use the NULL, #, and cnt (count) as indicated below.

<u>RANGE FORM</u>	<u>DESCRIPTION</u>
ADDR < null >	Range specifies the single memory location ADDR.
ADDR1-ADDR2	Range specifies the memory locations ADDR1 through ADDR2 inclusive.
ADDR/cnt	Range specifies cnt memory locations starting at location ADDR. NOTE: cnt is a decimal integer ≤ 255 .
#-ADDR	Range specifies the memory locations starting at the beginning of the previous range and ending at ADDR.
#/cnt	Range specifies cnt memory locations starting at the beginning or the previous range. NOTE: cnt is a decimal integer ≤ 255 .
< null>/cnt	Range specifies cnt memory locations starting at the address following the last address of the previous range. NOTE: cnt is a decimal integer ≤ 255 .
< null > -ADDR	Range specifies memory locations starting at the address following the last address of the previous range and extending to memory location ADDR.

For example, to display memory location 000 043 through 000 047, BUG-8 simply requires the user to type 43-47 followed by a blank (a blank is generated by using the console terminal space bar). For example:

```
B: 43-47 100 112 107 114 100
B:
```

```
B: /4 303 053 040 365
B:
```

NOTE: In the first example, the contents of memory locations 000 043 through 000 047 are displayed on the first line in byte format octal. The next four bytes (locations 000 050 to 000 053) are displayed when the command /4 is typed. The contents of these next four bytes are displayed as soon as a blank is typed after the /4.

If the first address specified is greater than the second address specified, an error message is generated and only the contents of the first memory location are displayed. The form of the error message is:

```
LWA>FWA
```

For example:

```
B: 47-43 LWA>FWA
100
B:
```

NOTE: If you attempt to enter a numerical address which does not fit the offset octal format, BUG-8 rejects the improper entry and sounds the console terminal bell. For example, the number 067777 does not fit the offset octal format; therefore, BUG-8 does not allow the second 7 to be entered.

Displaying Memory Contents

To display the values in the specified range and in the specified format, type a blank following the format and range fields. BUG-8 immediately executes the command. In the following examples, the contents of a number of locations, 000 043 to 000 070 in the PAM-8 ROM, are displayed in octal byte format, in octal word format, in decimal byte format, in decimal word format, in ASCII byte format, and finally in ASCII word format. NOTE: When all the bytes or words in the specified range cannot be displayed on the line, a new line is started. BUG-8 supplies the starting address of the new line.

```
B: 43-70 100 112 107 114 100 303 053 040 365 257 303 143 002 303 056
000062 040 076 320 303 235 001 303
```

```
B: F43-70 112100 114107 303100 040053 257365 143303 303002 040056
000063 320076 235303 303001
```

```
B: D43-70 064 074 071 076 064 195 043 032 245 175 195 099 002 195
000061 046 032 062 208 195 157 001 195
```

```
B: FD43-70 19008 19527 49984 08235 45045 25539 49922 08238 53310 40387
000067 49921
```

```
B: A43-70 @ J G L @ + C . >
B: FA43-70 J@ LG @ + C . >
```



Altering Memory — Decimal or Octal

To alter memory in decimal or octal formats, type an = after the format control and range fields. BUG-8 will then type the value of the first byte, or double byte if an F was used in the format control and follow this with a /. You can then type a new value if you want to change the contents of this location. If the contents of the location are not to be changed, or if sufficient new digits have been entered to complete the change, type a space or a carriage return.

If you type a space, BUG-8 offers the next byte (if there is one in the range) for alteration. If you type a carriage return, BUG-8 returns to the command mode.

In the following example, memory locations 60000 through 60031 are loaded with the octal values of the ASCII characters A through Z. NOTE: On the first three lines, the initial address is followed by the = sign, the current octal value in that memory location, and then a /. The octal value for the letter is entered following the slash. On the successive lines, a range of successive locations are opened and then changed to the sequentially ascending ASCII characters.

After the letters have been entered, the 26 memory locations are examined in byte format as ASCII characters. The 26 locations are then examined in word format as ASCII characters. Note that the second byte is treated as the most significant byte. Finally, the 26 locations are opened in byte octal format, using the # as the first address of the range.

```

B: 60000 = 000/101          (load A)
B: 60001 = 353/102          (load B)
B: 60002 = 341/103          (load C)
B: 60003/23=311/104 357/105 365/106 257/107 062/110 237/111 061/112
060012 303/113 274/114 061/115 315/116 230/117 062/120 012/121 376/122
042/123 302/124 135/125 057/126 120/127 131/130 023/131 046/132

B: A60000/26 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
B: FA#/26 BA DC FE HG JI LK NM PO RQ TS VU XW ZY
B: #-60031 101 102 103 104 105 106 107 110 111 112 113 114 115 116
060016 117 120 121 122 123 124 125 126 127 130 131 132
B;
```


The BACK SPACE and RUBOUT keys are not effective when you are entering memory locations. Values placed in memory are taken as modulus 256 numbers (if they are entered in byte format) or as modulus 65,535 numbers (if they are entered in full word format). Thus, if you make a mistake, simply type the correct value with enough leading zeros to cause the bad digit to be eliminated. For example, if byte 70,000 is to be set to 123 and the mis-type 125 occurs, it may be correctly entered as:

```
B: 70000=111/125123
B: 70000 123
```

NOTE: Only the three least significant digits are accepted for this byte location.

Altering Memory — ASCII Format

To alter memory in ASCII format, type an = after the format control (A for ASCII) and range fields. The processing is similar to decimal or octal format memory alterations. The contents of the opened locations should then be followed by a /. You can then enter the replacement character (or two characters if the word format is used). However, the space and the carriage return are considered to be ASCII character values. To exit the command prematurely, use the ESCAPE or CONTROL-C key to avoid altering a location.

```
B: A70000=/A
B: A70001=>/B
B: A70002=/C
B: A70003-70031=2/D /E /F /G /H /I /J /K" /L /M /N /OW/P#/Q /RZ/S /T
070024 /U /V8/W /X /Y/Z
B: 67374-67377 076 000 303 122
B: A-70031 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
B:
```



REGISTER COMMANDS

BUG-8 permits you to display the contents of all registers using octal, decimal, or ASCII, or to display the contents of individual registers using octal, decimal, or ASCII. In addition to displaying the contents of these registers, you can alter the various registers in any of the three modes. NOTE: If the F command is used in the format field, a register command is rejected, as register size is predetermined.

Displaying All Registers

To display the contents of all registers, enter a command of the form.

<FORMAT> <CNTRL-R>

BUG-8 displays the register contents in a specified format. NOTE: An M register is displayed in the all register command and can be specified in other commands. This register is the concatenation of the H and L registers. For example:

B: <CNTRL-R>

A=027 B=000 C=000 D=004 E=000 H=041 L=031 F=106 P=043324 M=041031
S=137377

B:

B: D<CNTRL-R>

A=023 B=000 C=000 D=004 E=000 H=033 L=025 F=070 P=09172 M=08473 S=24575
B:

B: A<CNTRL-R>

A= B= C= D= E= H=! L= F=F P=# M=! S=←

A Control R (CNTRL-R) should be typed after each command. However, no character is actually displayed. Also note that the ASCII display is not particularly meaningful unless printing ASCII characters are contained in the desired registers.

Displaying Individual Registers

To display the contents of any single register, use a command in the following format:

< FORMAT > REG < REG-NAME > < blank >



For example, to display the contents of register A, type:

```
B: REGA =101
B: DREGA =065
B: AREGA =A
B:
```

In the above example, the first line calls for the contents of register A to be displayed in octal format. In the second line, the contents of register A are displayed in the decimal format, and in the third line, the contents of register A are displayed in ASCII format. In the following example, the contents of the 16-bit register pair H and L, known as the M or memory register, are displayed in octal format.

```
B: REGM =041031
B: REGH =041
B: REGL =031
B:
```

Altering Register Contents

To alter the contents of a register, use a command in the following format:

```
< FORMAT > REG < REG-NAME > =
```

BUG-8 will then display the previous contents of the register (in the specified format octal, decimal or ASCII), followed by a /. It then accepts a new value if one is typed in. When you are using octal or decimal format, use a carriage return to close the entry or to skip the change. When you are using the ASCII format, type a single ASCII character to close the register. However, as the carriage return is a valid ASCII character, you must use ESCAPE or CONTROL-C to skip the change. The following examples demonstrate the altering of register contents.

B: REGA=102/103	(Change contents of A from 102 ₈ (ASCII B) to 103 ₈ (ASCII C).)
B: DREGA=067/066	(Change contents of A from 67 ₁₀ (ASCII C) to 66 ₁₀ (ASCII B).)
B: AREGA=B/C	(Change contents of A from ASCII B to ASCII C.)
B: REGA=103/ CR	(A carriage return skips the change.)
B: AREGA=C/ < CNTRL-C >	(A CONTROL-C skips the change.)
B: AREGA=C	(The location is unaltered.)

NOTE: The last three are examples of skipping the change (leaving the location unaltered).

EXECUTION CONTROL

One of the primary functions of BUG-8 is execution control. It lets you step through the program, one or more instructions at a time, while examining register and memory contents. In addition, complete breakpointing is available, permitting you to execute a number of instructions and then return to BUG-8 control to examine register and memory contents. Execution control is divided into the areas of single stepping, breakpointing, and the GO command.

Single Stepping

The form of the single step command is:

STEP ADDR/CNT

where ADDR is an offset octal address (or a null) and "CNT" is a decimal step count, ≤ 255 . If an address is not specified, BUG-8 starts stepping at the current PC-register address. When the instructions are completed, BUG-8 types the PC-register value and returns to the command mode. If an address is specified, BUG-8 starts stepping at the specified address and, when the instructions are completed, displays the terminating address value before returning to the command mode.

The following program increments the contents of memory location 055 100 each time the BC register pair is incremented from 000 000 to 027 000. This program is used to demonstrate a number of the execution control features of BUG-8.

<u>ADDRESS</u>	<u>LABEL</u>	<u>INSTRUCTION</u>	<u>OCTAL</u>	<u>COMMENT</u>
055 000	L1	LXI H	041	Point HL to 055 100.
055 001			100	
055 002			055	
055 003		MVI M	066	Load memory with 000.
055 004	L2		000	
055 005		INX B	003	Increment BC pair.
055 006		MOVA B	170	Load A with B.
055 007		CPI 027Q	376	Compare. Is B at 027 yet?
055 010			027	
055 011		JNZ L2	302	Jump back. Not 027 yet.
055 012			005	
055 013			055	
055 014		INR M	064	Passed 027. Increment mem

055 015	MOVA M	176	Load A with memory.
055 016	CPI 377Q	376	Compare. Is memory at 377 yet?
055 017		377	
055 020	MVI B	006	Reset B to 000.
055 021		000	
055 022	JNZ L1	302	Jump back. Not at 377 yet.
055 023		005	
055 024		055	
055 025	RST2	327	Memory is 377. Back to BUG-8.

NOTE: The RST2 instruction is used to return this program to BUG-8. When BUG-8 encounters an RST2 instruction, it checks the breakpoint table. If there is nothing set, it returns to BUG-8.

For example, to load the above program using BUG-8,

```
B: 55000-55025=164/041 055/100 376/055 110/066 312/000 252/003 055/170
055007 376/376 103/027 312/302 334/005 055/055 376/064 102/176 312/376
020/377 056/006 376/000 114/302 312/005 101/055 056/327
B:
```

Set the front panel of the H8 to display the BC pair. Zero the BC pair and step through the first 6 program steps using BUG-8 as in the following example. The BC pair will momentarily display 000 001 as the steps are executed.

```
B: REGB=053/000
B: REGC=132/000
B: STEP 55000/6
-P=055005-
```

BUG-8 returns the value of the PC once the first six steps are executed.

Breakpointing

BUG-8 contains several commands to set, display, and clear breakpoints in your program. Breakpointing permits you to execute portions of a program once (or a number of times if the portion of a program is in a loop). Breakpointing is especially useful in de-bugging programs which have a tendency to destroy themselves or obliterate the cause of the problem in the process of complete execution.

SETTING BREAKPOINTS

The breakpoint command is used to set a breakpoint. The form of the breakpoint command is:

```
BKPT ADDR1/CNT1, . . . . . ,ADDRn/CNTn
```



BUG-8 allows up to 6 breakpoints. They are entered in the breakpoint table within BUG-8, replacing any previously defined breakpoints at those addresses. No more than six breakpoints may be entered in the breakpoint table.

The CNT field may be used to specify the breakpoint repeat count. It is a decimal number in the range of 1 to 255. Using the breakpoint count means the breakpoint does not cause control to return to the monitor mode until the breakpoint is executed CNT-1 times. Thus, you may execute a loop a number of times prior to returning to the command mode via a breakpoint instruction. As noted, the Breakpoint Instruction executes CNT-1 times, without recognizing the breakpoint. The last time through the loop, the instruction at the breakpoint address is not executed. The breakpoint returns control to BUG-8. NOTE: If CNT is not specified, the value 1 is assumed.

For example, the program of the previous example is run with the H8 front panel displaying memory location 055 100.

```
B: 55100=001/000
B: BKPT 55015/6
B: GO 55000
-P=055015-
B:
```

NOTE: 055 100 is incremented by 6.

```
B: 55100=006/000
B: BKPT 55015/6, 55014/10, 55022/30
B: GO 55000
-P=055015-
B: GO
-P=055014-
B: GO
-P=055022-
B:
```

DISPLAYING BREAKPOINTS

To display the current status of the breakpoint table, use the breakpoint display command. BUG-8 can display the contents of the breakpoint table. The form of the breakpoint command is:

```
BKPT DSPLY
```

BUG-8 provides a listing of the current breakpoints in the form:

```
BKPT DSPLY ADDR1/CNT1 ADDR2/CNT2 . . . . ADDRn/CNTn
```

where ADDR is the address of the breakpoint instruction, and CNT are the loop counts remaining on the designated breakpoints. NOTE: When the breakpoint count is exhausted, it causes control to return to BUG-8. The breakpoint is removed from the breakpoint table, and nonexhausted breakpoints are saved.

For example:

```
B: 55100=036/000
B: BKPT 55015/6,55014/10,55022/30
B: BKPT DSPLY 055015/006 055014/010 055022 030
B: GO 55000
-P=055015
B: BKPT DSPLY 055014/004 055022/025
B: GO
-P=055014-
B: BKPT DSPLY 055022/021
B: GO
-P=055022-
B: BKPT DSPLY
B:
```

CLEARING INDIVIDUAL BREAKPOINTS

To clear an individual breakpoint, use the command

```
CLEAR ADDR1, . . . ,ADDRn
```

where ADDR1, . . . ,ADDRn specifies the address of the breakpoint to be removed from the table.

CLEARING ALL BREAKPOINTS

To complete clear all breakpoints from the breakpoint table, use the breakpoint clear command

```
CLEAR ALL
```

For example:

```
-P=040261-
B: BKPT 55012/10,55014/15,55020/20,55022/200
B: BKPT DSPLY 055012/010 055014/015 055020/020 055022/200
B: CLEAR 55014,55022
B: BKPT DSPLY 055012/010 055020/020
B: CLEAR ALL
B: BKPT DSPLY
B:
```



EXEC

The EXEC (execute) command is a combination of the GO and BKPT commands. The form of the EXEC command is:

```
EXEC SADDR-ADDR1, . . . . , ADDRn
```

where "SADDR" is the starting address for execution. If the starting address is omitted, execution starts at the current program counter register value. ADDR1 through ADDRn are the addresses of breakpoints to be set before execution. Thus, for example, to start at byte 055 000 and to execute to byte 055 015, the command is typed as:

```
B: EXEC 55000-55015  
-P=055015-  
B:
```

GO

Use the GO command to transfer control to your program. You can set breakpoints before via the BKPT command. The form of the GO command is:

```
GO [SADDR]
```

If you specify "SADDR," execution begins at this specified address. If you do not specify "SADDR," execution begins at the current value of the program counter register. For example, simple execution of the previous program is accomplished by

```
B: GO  
-P=0550250  
B:
```


TAPE HANDLING

BUG-8 offers three commands for tape handling. With these commands you can dump to a tape, load from a tape, or verify a tape.

DUMP

The DUMP command allows BUG-8 to write a file that is an image of the desired memory range. The operation is very much like the H8 front panel dump. The form of the DUMP command is:

```
DUMP ADDR1-ADDR2
```

ADDR1 and ADDR2 are specified in offset octal. Ready the tape transport before typing the carriage return after ADDR2. The contents of the Program Counter are also saved. Set the PC to the program entry point before you type a return.

LOAD

The LOAD command allows BUG-8 to read a file containing a memory image. The form of this command is:

```
LOAD
```

Once the load is complete, control is returned to BUG-8 for execution or other manipulations. When control is returned to BUG-8, the Program Counter is set to the entry point on the image tape. During a load a tape error may occur. These are explained in the "Tape Errors" section below.

VERIFY

The VERIFY command allows BUG-8 to verify a memory image file. The form of this command is:

```
VERIFY
```

BUG-8 reads the file without loading it and responds with

```
OK
```

if the CRC on the tape matches its computed CRC. If the CRCs do not match, a tape error is generated. (See "Tape Errors.")



Tape Errors

During a LOAD or VERIFY, one of two error messages may be generated. A checksum error is generated if the computed CRC for the record in question does not match the CRC at the start of the record. The form of the checksum error message is

```
CHKSUM ERR-IGNORE?
```

A Y in response to the question "ignore?" aborts the error message and the next consecutive record is accepted. NOTE: Ignoring the checksum error is not recommended unless there is no other way to recover the data. If a checksum error is flagged, the chances are very good the data in the designated record is faulty.

A sequence error (SEQ ERR) is generated if the file records are not in the proper sequence. For example, if two record numbers are not consecutive, an error is generated. The form of the sequence error is

```
SEQ ERR
```

Typing a space after the SEQ ERR message generates a tape error message and the entire file must be reread.

TERMINAL CONTROL CHARACTERS

BUG-8 recognizes a number of control characters. Some of these are valid when you are entering commands, and others are only valid when BUG-8 is printing information on the console terminal. You can make a control character by simultaneously depressing the appropriate key plus the CONTROL (CTRL) key. This is similar to depressing the SHIFT key along with a letter.

Input Control Characters

The following characters are only valid when you are entering commands.

BACKSPACE (CONTROL-H)

The BACKSPACE character causes the **last character** on the line to be discarded. A backspace code is sent to the CRT terminal so it can perform a cursor backspace, but the backspace is ignored by teleprinters, and some other printing terminals. If you attempt to BACKSPACE into column zero, a bell code is echoed noting this illegal operation. NOTE: Backspace can be changed when BUG-8 is configured. See “Appendix A,” in this section or Page 0-24 in the “Introduction” to this Software Reference Manual.

RUBOUT

The RUBOUT key causes the **current line** to be discarded. A carriage return/line feed is sent to the terminal. Once the RUBOUT is typed, the entire line may be re-entered. NOTE: Rubout can be changed when BUG-8 is configured. See “Appendix A,” in this section or Page 0-25 in the “Introduction” to this Software Reference Manual.

Output Control Characters

The following characters are only valid during output.

OUTPUT SUSPENSION AND RESTORATION, CNTRL-S AND CNTRL-Q

Typing Control S during an output suspends the output to the console terminal and suspends program execution. This command is particularly useful when you are using a video terminal, since you can use the Control S or suspend feature each time a screen is nearly filled and information at the top will be lost due to scrolling.

Typing Control Q permits BUG-8 to continue execution and output information to the terminal. The Control Q cancels the Control S function.



THE DISCARD FLAG, CTRL-O AND CTRL-P

Typing Control O toggles the DISCARD FLAG. This stops the output on the terminal but program execution does not halt until the program terminates. Typing a Control P (or typing Control O again) clears the discard flag. Control O is often used to discard the remainder of long outputs. Control P clears the discard flag.

COMMAND COMPLETION

When BUG-8 is in the command mode, each terminal keystroke is considered for validity. If the character belongs to no possible command, it is refused and the bell code is echoed to the terminal. If the command syntax allows only one next character, BUG-8 supplies and prints this character for the user.

In addition to simple syntax checking, BUG-8 also processes command range expressions as they are being entered. Should the user enter a range expression referring to nonexistent memory, BUG-8 refuses further entry and echoes a bell code. Thus, should apparently valid characters be rejected by BUG-8, it indicates that the command range expression may be invalid.



APPENDIX A

Loading Procedures

Loading From the Software Distribution Tape

1. Load the tape in the reader.
2. Ready the Tape transport.
3. Press LOAD on the H8 front panel.
4. A single beep indicates a successful load.
5. Install any optional patches.
6. Press GO on the H8 front panel.
7. Repeatedly type the space bar on your console until BUG-8 responds with:

```
HEATH/WINTEK TERMINAL DEBUGGER.
BUG-8 ISSUE # 02.03.00.
COPYRIGHT WINTEK CORP., 01/77
COPYRIGHT HEATH CORP., 06/78
```

8. Configure BUG-8 as desired, answering the following questions. Prompt each question by typing its first character on the console terminal keyboard.

```
•AUTO NEW-LINE (Y/N)?
•BKSP = 00008/
•CONSOLE LENGTH = 00080/
•HIGH MEMORY = 16383/
•LOWER CASE (Y/N)?
•PAD = 4/
•RUBOUT = 00127/
•SAVE?
```

9. Before executing SAVE, have the transport ready at the DUMP port.
10. To use BUG-8 directly from the distribution tape, type the return key at any time rather than a question prompt key. BUG-8 responds:

```
HEATH BUG-8 # 02.03.00
B:
```

BUG-8 is ready to use.

Loading From a Configured Tape

1. Load the tape in the tape transport.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. A single beep indicates a successful load.
5. Press GO on the H8 front panel. Then repeatedly press the space bar on your console.*
6. The console terminal should respond with:

```
HEATH BUG-8 # 02.03.00.  
B:
```

BUG-8 is ready to use in the configured form.

*NOTE: You must type the spaces to allow BUG-8 to determine the interface type and baud rate of your H8 Computer System. The number of spaces required depends on the type of console terminal, interface circuit board, and baud rate you are using.



APPENDIX B

BUG-8 Command Summary

Memory Commands

Memory display form:

FORMAT CONTROL range =

Memory Alter form:

FORMAT CONTROL range blank

<u>FORMAT</u>	<u>RESULT</u>
< null > < null >	byte octal
F < null >	word octal
< null > A	byte ASCII
FA	word ASCII
< null > D	byte decimal
FD	word decimal

Range

The range field consists of a beginning address and an ending address. You can specify addresses by using the appropriate offset octal integers; or you can use the NULL, #, and cnt (count) as indicated below.

<u>RANGE FORM</u>	<u>DESCRIPTION</u>
ADDR < null >	Range specifies the single memory location ADDR.

ADDR1-ADDR2	Range specifies the memory locations ADDR1 through ADDR2 inclusive.
ADDR/cnt	Range specifies cnt memory locations starting at location ADDR. NOTE: cnt is a decimal integer ≤ 255 .
#-ADDR	Range specifies the memory locations starting at the beginning of the previous range and ending at ADDR.
#/cnt	Range specifies cnt memory locations starting at the beginning of the previous range. NOTE: cnt is a decimal integer ≤ 255 .
< null >/cnt	Range specifies cnt memory locations starting at the address following the last address of the previous range. NOTE: cnt is a decimal integer ≤ 255 .
< null > -ADDR	Range specifies memory locations starting at the address following the last address of the previous range and extending to memory location ADDR.

Register Commands

All registers:

```
FORMAT CNTRL-R
```

Single register:

```
REG REG-NAME blank
```

Altering register:

```
REG REG-NAME=
```



Execution Control

Single stepping:

STEP ADDR/CNT

Breakpointing:

BKPT ADDR1/cnt 1, ,ADDRn/cntn (n-6)

Breakpoint display:

BKPT DSPLY

Clearing breakpoints:

CLEAR ADDR1, ,ADDRn
CLEAR ALL

GO:

GO (ADDR) (Starts at PC value if ADDR is not specified)

Execute:

EXEC SADDR-ADDR1, ,ADDRn (combines GO AND BKPT)

Tape Handling

DUMP ADDR1-ADDR2 ☞
LOAD ☞
VERIFY ☞

INDEX

ASCII Characters, 2-4
ASCII Format, 2-9
Altering Memory, 2-8
Altering Register, 2-11

Backspace, 2-19
Breakpointing, 2-13
Breakpoints, 2-15
Byte Format, 2-5

Clearing Breakpoints, 2-15
Command Completion, 2-21

Decimal Integers, 2-4
Displaying Breakpoints, 2-14
Dump, 2-17

Exec, 2-16
Execution Control, 2-12

Format Control, 2-4

GO, 2-16

Load, 2-17

Memory Commands, 2-4

Octal Integers, 2-4
Output Control Characters, 2-19

Range, 2-6
Register Commands, 2-10
Rubout, 2-19

Setting Breakpoints, 2-13
Single Stepping, 2-12

Tape Errors, 2-18
Tape Handling, 2-17

Verify, 2-17

Word Format, 2-5

Section 3

HEATH TEXT EDITOR

TED-8



TABLE OF CONTENTS

INTRODUCTION	3-3
EDITOR MODES OF OPERATION	3-4
The Command Mode	3-4
The Text Mode	3-4
THE COMMAND STRUCTURE	3-5
Range Expressions	3-5
The Verb	3-10
The Option Field	3-11
The Qualifier String	3-12
THE PARAMETER FIELD	3-12
THE COMMANDS	3-13
TERMINAL CONTROL CHARACTERS	3-28
Input Control Characters	3-28
Output Control Characters	3-28
Tape Errors	3-29
COMMAND COMPLETION	3-30
APPENDIX A	3-31
Loading From the Software Distribution Tape	3-31
Loading From a Configured Tape	3-32
APPENDIX B	3-33
Command Structure	3-33
Range Expression Forms	3-33
Line Expression Forms	3-33
Verb (Command) Forms	3-34
Option	3-36
Qualifier String	3-36
Parameter Field	3-36
Terminal Control Characters	3-36
APPENDIX C	3-37
INDEX	3-39

INTRODUCTION

The Heath H8 Text Editor (TED-8) converts your H8 computer system into a very sophisticated typewriter. This typewriter is not only capable of generating text, but also has powerful editing capabilities. With these capabilities, even a poor typist can create error-free text, organized as desired.

The principle purpose of TED-8 is the preparation of a special form of text for the Heath H8 Assembly Language program called source code. The format for assembly language source code is discussed in Section 4, the Heath Assembly Language Manual (HASL-8).

TED-8 is not restricted to producing source code for assembly language. It can also be used to prepare reports, write letters, and edit manuscripts.

Text is stored in a section of memory called the buffer. All memory not used by the text editor program is available as buffer. Editing can be done by command, referencing the desired line. When the buffer is full, text can be placed onto magnetic or punched paper tape files. Additional text can be read in from previously created magnetic or punched paper tape files, or can be placed in the buffer from the terminal keyboard. TED-8's tape handling capabilities allow it to edit large files on a piecemeal basis.

Slightly more than 4096 bytes of H8 memory are occupied by TED-8 and in addition to its program size, 200 additional bytes of working space are also required. The balance of H8 memory is available for use as buffer. An 8K memory, therefore, has approximately 4K of buffer space, which provides sufficient room for a well documented 300-line program. With less program documentation, more lines of code can be accommodated in the buffer. The same buffer accommodates approximately 175 lines of solid text, such as found in a report or letter.

TED-8 normally uses decimal notation when specifying a command range. Check the individual command description for details.

A compression technique is used in the Text Editor so large quantities of blanks will use very little buffer. This allows you to use sufficient blanks between the source code columns to make the source code easy to read, without using proportionate amounts of memory.

TED-8 has many unique features that are discussed in detail on the following pages. Some of these features are:

- 16 commands for text editing versatility.
- Terminal control of output and input operations.
- Command completion and command error analysis.



EDITOR MODES OF OPERATION

Two modes of operation called the "Command Mode" and the "Text Mode," are available in TED-8. These two modes distinguish between editing commands and text being entered into the buffer.

The Command Mode

The command mode can be subdivided into three areas: input commands, output commands, and editing commands. You execute all commands by typing the appropriate command on the terminal, and follow this with a carriage return; this will be indicated throughout this reference Manual by the symbol CR . In actual use, no symbol is printed when a carriage return is typed on the terminal, as the carriage or cursor simply moves to the first column of the next line. In the command mode, the prompt character - - (a double dash) appears in the first two columns.

The Text Mode

In this mode you can add text to the buffer from the terminal keyboard, the normal source for most text. But once a source file has been created, it can be stored on a tape file. Later, you can use this tape file as a source of text to be added to the buffer.

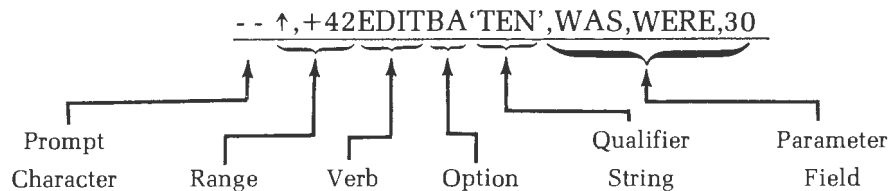
Type the ESCAPE (ESC) key when you wish to return from the text mode to the command mode. This performs two functions. First, it deletes the line of text which it was on when the key was struck. Second, it returns the Text Editor to the command mode. To preserve the last line of text, type a carriage return to generate a new line before striking the ESCAPE key. If the console terminal does not have an ESCAPE key, use CONTROL-C.

THE COMMAND STRUCTURE

The basic commands for TED-8 have the following form:

```
[<range>] [<verb>] [<option>] [<qualifier string>] [<parameters>]
```

The **range** indicates what lines in the buffer the command affects; the **verb** is the basic command. The **option** permits you to view the line before or after (or both) the command is executed. The **qualifier string** limits the command to those lines containing a given string; and the **parameters** (parameter field) contains specific instructions for some commands. For example, the command



is read as follows:

The lines starting with the first line (↑) and ending with the 43rd line (+42) are to be edited (EDIT). The EDIT command replaces one string with another. The option (BA) indicates that you wish to view the lines before and after editing. The affected lines are limited to those containing the string "TEN" (the optional qualifier string). And in this particular case, the parameter field specifies the old and new strings and the count. The word WAS is to be replaced by the word WERE, a maximum of 30 times. NOTE: TED-8 strings used in the range expression or qualifier strings must be enclosed in a single quote (') (apostrophe).

Range Expressions

The range expressions define the buffer lines on which the command is to operate. They may define a single line, or two expressions may be used to define the range over which the command works.

A range expression may consist of:

- A line expression.
- A two-line expression.
- A null.
- A blank.
- An equal sign.



These different range expressions are explained below.

NOTE: Text must be in the buffer before a range expression will be accepted. A bell code will sound as you try to complete the range expression. To use the following examples, configure TED-8 (see Page 3-28) and use the INSERT command (see Page 3-14).

SINGLE-LINE EXPRESSIONS

Either one of the following two line expressions may be used to define a single line:

<u>THIS SYMBOL</u>	<u>DEFINES</u>
↑	The first line.
\$	The last line.

TED-8 is always pointing to some line of text. Once you have explicitly pointed to a line by referencing the first line (↑) or the last line (\$), you may make future line references by referencing to the current line pointer. NOTE: Once a DELETE has been executed, the current line pointer is reset and you must explicitly reference a line to reestablish the line pointer.

- A. + count. A plus (+) followed by a decimal number (n) refers to the nth line past the current line pointer.
- B. – count. A minus – (n) refers to the nth line preceding the current line pointer.
- C. + ‘string’. The + ‘string’ refers to the first line in the text buffer which contains the designated ‘string’ after the current line pointer.
- D. – ‘string’. The – ‘string’ refers to the first line in the buffer, preceding the current line pointer, containing the indicated ‘string’.

For example, assume the buffer contains:

```
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
--
```

TED-8 responds as follows to the range commands.

```
--↑PRINT
THIS IS THE FIRST LINE
--$PRINT
THIS IS THE LAST LINE
--↑+2PRINT
THIS IS THE THIRD LINE
--↑+'FLEECE'PRINT
ITS FLEECE WAS WHITE AS SNOW
--$-1PRINT
THE LAMB WAS SURE TO GO
--$-'EVERYWHERE'PRINT
AND EVERYWHERE THAT MARY WENT
--
```

MULTIPLE LINE EXPRESSIONS

Use a two-line expression when you want to define a group of lines to be operated on by the command. Use the comma as a delimiter to separate the start line from the stop line. The symbols ↑, \$, + count, - count, + 'string', and - 'string' have the same meaning as they do with a single-line command. NOTE: A wide range of combinations may be used to identify the first and last lines of a two-line expression.

For example, you could print the contents of the previous buffer using these commands:

```
--↑,↑+3PRINT
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW

--↑+2,+3PRINT
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT

--↑+'FLEECE',+1PRINT
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE

--$-'THAT',+'SURE'PRINT
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO

--$-'THAT',+2PRINT
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
--
```

THE BLANK

When the verb is preceded by a single blank (space), the range is the entire buffer. For example, printing the entire buffer is accomplished by:

```
-- PRINT
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
--
```

Note the blank (space) between the prompt character (--) and the word PRINT.
The blank is created by typing the space bar on the console terminal.

THE NULL

The NULL expression (the absence of any character) sets a single-line range at the first line of the previous command.

For example:

```
--↑+2,+'FIFTH'PRINT
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
--PRINT
THIS IS THE THIRD LINE
--
```

Note that there is no blank (a NULL) between the prompt (--) and the word PRINT.

```
--↑+2,+1PRINT
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
--, +2PRINT
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
--
```

NOTE: In this example, the NULL starts the range at the first line of the previous range, and the ,+2 directs TED-8 to print two additional lines.



THE EQUAL (=)

The = expression sets the range to the range of the previous command. For example:

```
--$- 'MARY' , + 'GO' PRINT  
  
AND EVERYWHERE THAT MARY WENT  
THIS IS THE SEVENTH LINE  
THE LAMB WAS SURE TO GO  
--=PRINT  
AND EVERYWHERE THAT MARY WENT  
THIS IS THE SEVENTH LINE  
THE LAMB WAS SURE TO GO  
--
```

Note that the command =PRINT causes the same lines from the previous command range to be printed in the second half of the example.

The Verb

The verb specifies the action to be taken by the Editor. For example, the command PRINT or the command EDIT are verbs within the text editor's vocabulary.

All verbs are command completed. As soon as the Text Editor receives enough characters to know that only one command is possible, it prints the balance of the command without any additional keys being struck.

The verb will be refused if it is not valid for the current TED-8 condition. For example, an attempt to PRINT an empty buffer is meaningless and the PRINT verb will be rejected.

For example, when you type the P key, the Text Editor knows that no other command begins with P. Therefore, the P is immediately followed by RINT. However, if you type the N, it does not know if the command NEWIN, NEWOUT, or NEXT is to be used. So the Editor prints NE and waits for a W or X. If it receives a W, it then waits for an I or an O. It completes these by filling in the N or UT. If it receives an X following the E, it then prints the T. A complete list of all of the command verbs, and their specific actions and limitations follows in "The Commands" (Page 3-14).

The Option Field

The option field contains characters which let you view the line to be worked on, and/or the line after it has been worked on.

As its name implies, it is completely optional. You may insert any one of the following three option forms, or none at all.

1. B The BEFORE option displays the line **before** the command is executed.
2. A The AFTER option displays the line **after** the command execution.
3. BA This is a combination of the previous two commands. The line is displayed **before** and **after** command execution.

For example, presume that the buffer erroneously contained

```
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS RED AS SNOW
THIS IS THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
--
```

It may be edited to read correctly by

```
--↑+'RED'EDITBA,RED, WHITE, 1
ITS FLEECE WAS RED AS SNOW
ITS FLEECE WAS WHITE AS SNOW
--
```

Note that after the command EDIT, the options B & A are used to check the work.

NOTE: There are certain times when the command renders the option meaningless. DELETE BA, for example. The AFTER portion of the command is meaningless, as the line (or lines) cannot be displayed after deletion.



The Qualifier String

The qualifier string is a further restriction upon the range expression. It is completely optional. If it is not indicated, it is not used.

The range expression may indicate work over a certain number of sequential lines, starting with a given line. The qualifier string further limits these lines to those within the range containing the string specified in the qualifier field. This string is enclosed in single quotes (') and contains all normal ASCII characters with the exception of the single quote (').

For example, to print the entire buffer (of the previous example), but only those lines with the string 'line':

```
-- PRINT'LINE'
THIS IS THE FIRST LINE
THIS IS THE THIRD LINE
THIS IS THE FIFTH LINE
THIS IS THE SEVENTH LINE
THIS IS THE LAST LINE
```

The Parameter Field

This is a special field used with the EDIT, TAB, NEWIN, and NEWOUT commands. These commands require additional operating information, which is placed in the parameter field. The nature of this information depends upon the command used. The exact format is discussed under those commands.



THE COMMANDS

The following paragraphs give a complete description of each of the command verbs. Examples of many commands are also given to demonstrate some of the combinations of range expressions, options, qualifier strings, and parameter fields (if required) that may be used with this command. NOTE: All possible combinations of range expressions, options, qualifier strings, and parameter fields are not given. You must review the section for each of these expressions to determine all possible command structures.

INSERT

The INSERT command places TED-8 in the text mode, and is used to add text to the buffer from the console keyboard. This command adds text to the buffer on the next line following the first line of the range expression. The second line of the range expression is meaningless. Also, if the range expression is given as a line minus a count, the appending point is still the next line following the indicated line. The option, qualifier string, and parameter fields are ignored for the INSERT command. The range command blank INSERT is a special case that is used to insert a line before the first line in the buffer.

For example:

```
-- PRINT
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
--

--$INSERT
THIS IS AN ADDITIONAL LAST LINE
<CNTRL-C>
--

--↑+'FIFTH'INSERT
THIS IS A NEW LINE INSERTED AFTER THE FIFTH LINE
--

-- PRINT
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
```



```
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
THIS IS A NEW LINE INSERTED AFTER THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
THIS IS AN ADDITIONAL LAST LINE
--
```

NOTE: Use the ESCAPE key (or the CONTROL-C) to return to the command mode. This will cause the current line to be erased. If you strike the ESCAPE key after inserting a partial line, that partial line will be lost. Therefore, to preserve the last line of inserted text, type a carriage return (this will create a new blank line in the text buffer) prior to typing the ESCAPE key.

REPLACE

This command causes the eligible line(s) in the command range to be replaced. Once the executing carriage return is typed, the terminal cursor moves to the start of the first line to be replaced. Type the replacement lines one at a time. Tabs, backspaces, and rub-outs may be used as part of the replacement text. However, typing a carriage return signifies replacement of another line within the command range.

After the lines indicated by the range expression have been replaced, TED-8 reverts to the command mode. Typing ESCAPE (or CONTROL-C) returns the editor to the command mode, erasing the current line.

The option and qualifier strings may be used. There is no parameter field for the REPLACE command.

For example:

```
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
THIS IS A NEW LINE INSERTED AFTER THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
THIS IS AN ADDITIONAL LAST LINE
--
```



```
--↑+5REPLACEB
THIS IS A NEW LINE INSERTED AFTER THE FIFTH LINE
THIS LINE IS REPLACED
--
```

```
--↑+6,+'LAST'REPLACE'LINE'
THIS REPLACES THE OLD SEVENTH LINE
THIS REPLACES THE OLD LAST LINE
--
```

```
-- PRINT
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
THIS LINE IS REPLACED
AND EVERYWHERE THAT MARY WENT
THIS REPLACES THE OLD SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS REPLACES THE OLD LAST LINE
THIS IS AN ADDITIONAL LAST LINE
--
```

Note: Only lines containing the string 'line' are replaced in the second example, as the string 'line' is used as a qualifier.

DELETE

The DELETE command causes the eligible line(s) in the command range to be deleted. You may use the option B; however, the option A is meaningless. You may also use the qualifier string. There is no parameter field accompanying the DELETE command. You may not delete the entire buffer. To do this, use the BLITZ command.

For example:

```
-- PRINT
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
THIS LINE IS REPLACED
AND EVERYWHERE THAT MARY WENT
THIS REPLACES THE OLD SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS REPLACES THE OLD LAST LINE
THIS IS AN ADDITIONAL LAST LINE
--
```

```
--$DELETE
--$PRINT
THIS REPLACES THE OLD LAST LINE
--

--↑,$-1DELETED
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
THIS LINE IS REPLACED
AND EVERYWHERE THAT MARY WENT
THIS REPLACES THE OLD SEVENTH LINE
THE LAMB WAS SURE TO GO
--PRINT
THIS REPLACES THE OLD LAST LINE
--
```

EDIT

Use the EDIT command to replace one string with another. The string to be replaced and the new string are given in the parameter field of the EDIT command. The parameter field of the EDIT command takes the form:

```
<delimiter> old string <delimiter> new string <delimiter> <count>
```

The **delimiter** is an arbitrary delimiting character. (It may not be a carriage return). The **count** is a decimal count showing the number of replacements to be made. NOTE: Only ONE replacement is made PER LINE. If the count is left null, it defaults to one. If an asterisk (*) is substituted for the count, the value 65,536 is assumed.

The EDIT command makes full use of all range expressions, options, and qualifier strings; and as noted, a specific parameter field.

The following is an example of a text buffer prior to using an EDIT command, and text buffer after the EDIT command is executed.

```
-- PRINT
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
--
```



```

--△EDIT/LINE/OF MANY LINES/5
--△PRINT
THIS IS THE FIRST OF MANY LINES
MARY HAD A LITTLE LAMB
THIS IS THE THIRD OF MANY LINES
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH OF MANY LINES
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH OF MANY LINES
THE LAMB WAS SURE TO GO
THIS IS THE LAST OF MANY LINES
--

```

NOTE: The use of an arbitrary delimiting character permits you to choose a delimiting character not contained in the old or new strings. For example, if a comma (,) is used as a delimiter, it may not appear in either string. If either string contains a comma, you can use a slash (/) as a delimiter.

PRINT,nnn

The PRINT command causes the eligible line(s) in the command range to be printed. This is the most common method for viewing the text buffer. All forms of the range expressions are utilized. The option commands have no effect. You may use the qualifier string to limit the range expressions.

The form of the PRINT,nnn command is:

```
PRINT,nnn
```

where nnn (the parameter field is greater than 9) specifies the size of the page (the number of lines per page). If no value is specified, PRINT does not recognize page boundaries. The actual number of lines printed is nnn-6. There are six blank lines printed between pages. The following command, for example, causes the contents of the buffer to be printed.

```

-- PRINT,32
POSITION PAPER. THEN TYPE ␣

```

Enter a carriage return after the paper is positioned. Each page contains 26 lines of text and then skips six lines before printing more text.

If the PRINT, nnn command encounters CTRL-L (form feed) in the text, it will skip to the top of the next page and continue printing. If the page size (nnn) is not specified and a CTRL-L is encountered, it will only skip six lines and continue printing.

NOTE: While the PRINT command is executing, you may type control characters, such as CTRL-S, to aid in viewing the text buffer. See "Terminal Control Characters," on Page 3-28.

TAB

The TAB command sets tab stops for entering text. This is especially useful when you are entering source code for assembly programs so the label field, op code field, operand field, and comments field can be clearly separated. TED-8 converts a TAB into an appropriate number of blanks and inserts these into the text. The TAB character itself is not inserted into the text. NOTE: Blanks inserted by a TAB do not use proportional quantities of buffer space. The TAB command uses the parameter field to set the tab stop up to a maximum of six tabs. The format of the parameter field is:

TAB, 1, , n

The numbers 1 through n specify the columns, by number, which the TAB is to stop. Previously-set tab stops are discarded once the TAB command is called.

Once the tabs are set, you may use the TAB key to space over to the next tab stop. (Some console terminals do not have a TAB key; they perform the tab function when you simultaneously press the CTRL and the I keys.) An audio alert is sounded when you exceed the number of tabs available. The tab command does not use the range field, option field, or qualifier field. The following example shows you how to set the tabs and use the TAB key.

NOTE: The (TAB) indicates that the user typed a tab. The cursor automatically moves over to the beginning of the new columns. The user did not type in the blanks between columns. This was performed by the TAB command.

```
--TAB, 10, 20, 30, 40
--INSERT
*      DETERMINE MEMORY LIMIT
INIT1  TAB  MOV  (TAB)      M, A (TAB)      MOVE BYTE
(TAB)      DAD  (TAB)      D (TAB)          INCREMENT TRIAL ADDRESS
(TAB)      MOV  (TAB)      A, M
(TAB)      DCR  (TAB)      M
(TAB)      CMP  (TAB)      M
(TAB)      JNE  (TAB)      INIT1 (TAB)      IF MEMORY CHANGED

INIT2  TAB  DCX  (TAB)      H
(TAB)      SPHL (TAB)      (TAB)          SET STACK POINTER=MEMORY LIMIT-1
(TAB)      PUSH (TAB)      H (TAB)          SET *PC* VALUE ON STACK
(TAB)      LXI  (TAB)      H, ERROR
(TAB)      PUSH (TAB)      H (TAB)          SET: RETURN ADDRESS:
(TAB)      MVI  (TAB)      A, UMI.1B+UMI, L8+UMI.16X
(TAB)      OUT  (TAB)      OP.TPC (TAB) SET 8 BIT, NO PARITY, 1STOP, X16
--
```



BLITZ

The BLITZ command discards all text in the working buffer. Because of the drastic action this command takes, BLITZ followed by a carriage return results in the question SURE? A Y in response to SURE? proceeds with BLITZ. If you inadvertently type a B, thus causing a BLITZ, you may abort the BLITZ by typing an N or any character except Y. The range option, qualifier and parameter fields are ignored in a BLITZ command. The BLITZ command **does not** reset tab stops.

USE

The USE command displays the number of lines in the command range, the number of memory bytes currently used, and the number of free bytes. The USE command replies giving three values:

1. A line count. The number of lines within the command range. Type USE to display the total number of lines presently used within the buffer. A null USE results in a single line indication.
2. A byte count. The number of bytes used by the entire working buffer, not simply the lines within the command range.
3. A bytes free count. This is the number of remaining bytes in memory.

You can use the range expression and qualifier strings, but they have little meaning. There is NO parameter field. The following example demonstrates the USE command. The H8 employed in this example has 16 K of memory with the following text:

```
*          DETERMINE MEMORY LIMIT

INIT1  MOV   M,A      MOVE BYTE
        DAD   D        INCREMENT TRIAL ADDRESS
        MOV   A,M
        DCR   M
        CMP   M
        JNE   INIT1   IF MEMORY CHANGED

INIT2  DCX   H
        SPHL           SET STACK POINTER=MEMORY LIMIT-1
        PUSH  H        SET *PC* VALUE ON STACK
        LXI   H,ERROR
        PUSH  H        SET:RETURN ADDRESS:
        MVI           A,UMI.1B+UMI,L8+UMI.16X
        OUT           OP.TPC SET 8 BIT,NO PARITY,1STOP,X16

--
```


NOTE: TED-8 requires some room for work space. It refuses to allow more text when there are still 200 free bytes. These 200 bytes are its work space.

```
-- USE
  LINES=00017
  USED=00338
  FREE=11537
--
```

NEWIN

The NEWIN command opens a tape file for reading. It permits you to search a tape that contains a number of files for a particular named file to be used by the Editor. (It is not necessary to have an input tape file to do editing. You may create a new file in the buffer by using the INSERT command.) The name of the text file to be opened is contained in the NEWIN command parameter field. The parameter field for the NEWIN command is in the form:

<delimiter> <name> <delimiter>

Be sure the tape unit is made ready before you type the carriage return. TED-8 will then scan the tape until it finds the named text file.

Note: A file is acceptable if the leading characters in its name match all the characters supplied in the NEWIN parameter field. In other words, the full name need not be supplied. Thus, supplying null as a name (simply two delimiter characters together) allows a match on the first text file found.

After the label record is found, the tape stops before the first block of text. TED-8 responds by indicating the name of the file that has been found. Use a FILL or READ command to input text from this file. The NEWIN command does not use range expressions, options, or qualifier strings.

NOTE: If an input file is opened but not read to the end-of-file record, the NEWIN command asks for a go-ahead prior to searching for a new file. The reply Y, to SURE?, instructs NEWIN to proceed to find the new file.

For example:

```
--NEWIN"USR"

OLD "IN" NOT FINISHED. SURE?YFOUND USR PROGRAM FOR BASIC #1
--
```



FILL

This command fills the buffer with text from an input file opened by the NEWIN command. FILL causes successive records of text to be read from the input tape until:

1. An end-of-file is read; or
2. Less than 512 bytes (1 block) of free buffer space is left.

The FILL command ignores range, expressions, options, and qualifier strings. It does not use the parameter field. When the prompt returns, FILL is complete.

READ

The READ command is used to input one **record** of text from the tape. If insufficient buffer room exists to hold a record of text, an error message is given. In this situation, you must first empty the buffer before the READ command can be executed. The range option expression and qualifier fields are ignored. The READ command does not use the parameter field.

Text inputted by the READ command is appended to the working buffer. NOTE: The READ command differs from the FILL command, as the READ command only inputs one record from the tape. When the prompt returns, READ is complete.

NEWOUT

The NEWOUT command is used to open a new output file. The file name is supplied in the parameter field, in the same manner as in the NEWIN command. The form of the parameter field is:

`<delimiter> <name> <delimiter>`

The output tape should be readied before you type the carriage return. TED-8 will write a label record on the tape and then stop it, leaving it ready for text. Use the WRITE, FLUSH, or SAVE commands, as appropriate, to write text onto the tape.

The NEWOUT command does not use range expressions, options, or qualifier strings.

If you use NEWOUT without closing a previously opened file, NEWOUT responds with SURE? A reply of Y permits NEWOUT to write the label for the new file without closing the previous file. **WARNING:** This procedure results in a partially complete file being left on the output tape. The FLUSH or STOP OUTPUT commands are used to write an end-of-file.

WRITE

The WRITE command outputs text from the buffer onto tape. It starts at the top of the buffer and continues to the first line of the command range. Thus, the command

```
$WRITE
```

writes the entire buffer.

This command uses range expressions, options, and qualifier strings. It has no parameter field. **NOTE: After lines are written, they are deleted from the buffer.**

FLUSH

The FLUSH command is used when editing is complete. It causes the working buffer to be written to the output tape, and then any remaining text on the input tape is copied over to the output tape without modification. The FLUSH command then writes an EOF (end-of-file) record on the output tape. This EOF record is needed for subsequent reading of the output file.

NEXT

The NEXT command performs the following sequence:

```
$WRITE  
FILL
```

This command causes all the text in the buffer to be written to the output tape, and then refills the buffer from the input tape. This command is particularly useful when you are generating long tape files or when you perform minor editing on long tape files. The NEXT command does not use a range expression, options, or qualifier strings. It has no parameter field.

SAVE

The SAVE command outputs text from a working buffer onto tape. The SAVE command starts at the first line in the buffer and continues to the first line of the command range. Thus the command

```
$SAVE
```

saves the entire buffer.



The SAVE command uses range expressions, options, and qualifier strings. It has no parameter field. NOTE: After the buffer is saved, you can use a VERIFY to be sure the tape record contains the desired information before you erase the buffer. SAVE is identical to WRITE except the buffer is not deleted after SAVE outputs to the tape. NOTE: The command SAVE (no range field) will only save one line.

STOP

There are two forms of the STOP command. They are:

STOP INPUT

and

STOP OUTPUT

The STOP INPUT command sets the flag in TED-8, which indicates that an EOF has been read. Once TED-8 executes a STOP INPUT, the previously opened input file is presumed closed.

The STOP OUTPUT command instructs TED-8 to write an EOF record on the current output tape. STOP OUTPUT is used following a \$SAVE or \$WRITE command to close the output file.

SEARCH

The SEARCH command scans through a text file for a given character string. The desired character string is specified in the qualifier field. This command begins the scan at the first line in the command range and continues to the end of the buffer (regardless of the last line in the command range). If the given character string is still not found, the buffer is written onto the output tape and more text is added from the input file. The SEARCH stops when an EOF is read, or when the string is found.

When the string is found, the command range is set to that line. Subsequent commands can reference the line containing the desired string by using an equal (=) for the range expression. The SEARCH command uses the range expression, but does not use option or parameter fields.

For example:

```
-- PRINT
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
```

```
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
--
```

```
--SEARCH"FIFTE"
--=EDITBA,FIFTE,FIFTH,
THIS IS THE FIFTE LINE
THIS IS THE FIFTH LINE
--
```

NOTE: The given character string is enclosed in double quotes (") not single quotes (').

VERIFY

The form of the VERIFY command is:

```
VERIFY n
```

where n is the number of records to be verified. If no value is specified, VERIFY verifies to the last record written and stops the tape (a record written by a SAVE for example). In either case, VERIFY stops when an EOF is read.

The VERIFY command is used to check a mass storage record without loading it in the buffer. The VERIFY command checks the sequence and CRC of each record in the file. If a valid ASCII data file is found preceded by a label record, VERIFY responds with

```
RECORD #000 OK
RECORD #001 OK
RECORD #002 OK      etc.
--
```

NOTE: Record #000 is the label record that contains the file name, etc. All other records are compressed ASCII records containing text. The number of records depends on the length of the file.

XPORT

The XPORT command lets you specify a device other than the console terminal for I/O operations. This alternate device can be a line printer, a paper tape reader/punch, or another console terminal. These alternate I/O devices must be connected to your H8 computer system with either an H8-4 or H8-5 Serial Interface Card. The form of the XPORT command is:

```
XPORT CR
```



TED-8 will respond to the carriage return by printing:

```
PORT <340Q>?
```

The prompt indicates that the Text Editor needs to know what port you connected the external device to. The "<340Q>" is the default value. This means that TED-8 assumes you wish to transport the data to the line printer (H14), normally configured at port 340Q. If you are attached to an H14, simply enter a carriage return for the default value. The "Q" is the numbers "postradix", a way of telling TED-8 that the number is in base eight (OCTAL). Valid postradixes recognized by TED-8 are:

	Base 10	(decimal -- blank)
Q	Base 8	(octal -- Q)
B	Base 2	(binary -- D)

For instance, the numbers 250 and 372Q refer to the normal console port assignment terminal when you are using the H8-5 Serial Interface Card.

After you respond to the "PORT <340Q>?" prompt, the Text Editor must know what baud rate you are using. You respond to the next prompt by entering the correct baud rate for the device.

```
BAUD RATE <H8-5>?
```

The default value is an H8-5 Serial Interface Card. This means you only need respond by entering a carriage return when the alternate device is attached using an H8-5. However, a device connected to an H8-4 requires that you always specify a baud rate up to 9600 baud for each peripheral device.

NOTE: The XPORT command does not move data, but only reassigns port addresses for the XINSERT and XPRINT commands.

The XPORT command lets you transfer data between devices using a standard ASCII serial format. This includes all ASCII terminals, such as the Model 33 teletype.

You may also interface the device to a paper tape reader using the H8-2 Parallel Interface circuit board. In the case of an H8-2, the baud rate is meaningless and you must enter a carriage return in response to the "BAUD RATE?" prompt.

The XPORT command remains in effect until you use another XPORT command to reassign (return) I/O operations.

XINSERT

The XINSERT command lets you enter text from an alternate device. The device must have been previously configured by the XPORT command. The text is not echoed on the alternate device but is printed on the console. This feature is useful when you are loading full, not compressed, ASCII text, such as that found on paper tapes created by a teleprinter.

Use the following sequence of commands to read an ASCII paper tape from a device attached at port location 20Q with an H8-2 Interface Card:

```
--XPORT  
PORT <340Q>? 20Q  
BAUD RATE  
--XINSERT
```

Always activate the paper tape reader before you press the carriage return.

NOTE: If you press the ESCape or CTRL/C key, control will return to the console.

XPRINT,nnn

The XPRINT command is identical to the PRINT command except that output is directed to a different port location. This port location must have been previously configured by the XPORT command. The XPRINT command is useful for diverting output to a line printer.

In the following example, the contents of the buffer are directed to an H36 line printer. The printer is connected to port location 374 with an H8-5 Serial I/O Card.

```
--XPORT  
PORT <340Q>? 374Q  
BAUD RATE <H8-5>?  
-- XPRINT
```



TERMINAL CONTROL CHARACTERS

TED-8 recognizes a number of control characters. Some of these are valid when you are typing into the TED-8, and others are only valid when TED-8 is printing information on the terminal or line printer. You can make a control character by simultaneously depressing the appropriate key plus the CONTROL (CTRL) key. This is similar to depressing the SHIFT key along with a letter.

Input Control Characters

The following characters are only valid when you are inputting.

BACKSPACE (CONTROL-H).

The BACKSPACE character causes the **last character** on the line to be discarded. A BACKSPACE code is sent to the terminal so CRT terminals can perform a cursor BACKSPACE. If you attempt to BACKSPACE into column zero, a bell code is echoed noting this illegal operation. BACKSPACE is valid in the command and text modes.

BACKSPACE can be changed when the Text Editor is configured. See “Appendix A,” or “Product Installation” on Page 0-23 of the “Introduction” to this “Software Reference Manual.”

NOTE: The BACKSPACE is ignored by teleprinters and some other printing terminals.

RUBOUT

The RUBOUT key causes the **current line** to be discarded. A carriage return/line feed is sent to the terminal. Once the RUBOUT is typed, the entire line may be re-entered. RUBOUT is valid in the command and text modes. NOTE: RUBOUT can be changed when the test editor is configured. See “Product Installation” on Page 0-23 of the “Introduction to this Software Reference Manual.”

TAB (CONTROL I)

The TAB character is valid only when you are adding text to the buffer from the keyboard. It is used with the INSERT and REPLACE commands. Typing TAB (CONTROL I) causes the cursor to advance to the next tab column. If there is no next tab column, a bell code is echoed. TED-8 converts TAB's into the equivalent number of blanks and stores these in compressed form.

Output Control Characters

The following characters are only valid during execution of the PRINT command.

OUTPUT SUSPENSION AND RESTORATION CTRL-S AND CTRL-Q

If you type CONTROL S during an output, it suspends the output to the terminal by setting the suspend flag.

Typing CONTROL Q permits TED-8 to continue execution and outputting information to the terminal. The CONTROL Q clears the suspend flag.

THE DISCARD FLAG, CTRL-O AND CTRL-P

If you type a CONTROL O, it toggles the DISCARD FLAG. This stops output on the terminal but does not halt program execution until the program terminates. Typing CONTROL O again toggles the discard flag, resuming output. Type a CONTROL P to clear the discard flag. CONTROL O is often used to discard the remainder of long listings and other similar outputs.

Tape Errors

During a NEWIN, FILL, READ, or VERIFY, either one of the following two error messages may be generated:

```
SEQ ERR
CHKSUM ERR.
```

A sequence error (SEQ ERR) is generated if the file records are not in the proper sequence. For example, if two record numbers are not consecutive, an error is generated. The form of the sequence error is

```
SEQ ERR
```

Typing a blank after the SEQ ERR message generates a tape error message and the entire file must be reread.

A checksum error (CHKSUM ERR) is generated if the computed CRC for the record in question does not match the CRC recorded at the start of the record. The form of the checksum error message is

```
CHKSUM ERR    IGNORE?
```

A Y in response to the question “ignore” aborts the error message and the next consecutive record is processed. NOTE: Ignoring the checksum error is not recommended unless there is no other way to recover the data. If a checksum error is flagged, the chances are very good that the data in the designated record is faulty, and may cause TED-8 to crash.



COMMAND COMPLETION

When TED-8 is in the command mode, each terminal keystroke is considered for validity. If the character belongs to no possible command, it is refused and the bell code is echoed to the terminal. If the character is accepted and the command syntax allows only one next character, TED-8 supplies and prints this character for you.

In addition to simple syntax checking, TED-8 processes command range expressions as they are being entered. If you should enter a range expression referring to nonexistent lines, TED-8 refuses further entry and echoes a bell code. Thus, should valid characters be rejected, it indicates that the command range expression is invalid. For example, if you should attempt to type

```
-↑+'TUESDAY'
```

and TED-8 refuses the “P” in PRINT, it means that it found no line containing Tuesday. NOTE: This is done before the command was executed, so you can use a backspace to eliminate the TUESDAY and replace this string with a string valid for the text. NOTE: If no information exists in the text buffer, TED-8 will not accept commands which need text to be valid. For example, the command PRINT is invalid when no text exists in the text buffer, as there is nothing to print.

APPENDIX A

Loading Procedures

Loading From the Software Distribution Tape

1. Load the tape in the reader.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. A single beep indicates a successful load.
5. Install any optional patches.
6. Press GO on the H8 front panel.
7. Repeatedly type the space bar on your console until TED-8 responds with:

```
HEATH/WINTEK H8 EDITOR
TED-8 # 3.03.00.
COPYRIGHT WINTEK CORP., 01/77
COPYRIGHT HEATH COMPANY, 03/78
```

8. Configure TED-8 as desired, answering the following questions. Prompt each question by typing its first character on the console terminal keyboard.
 - AUTO NEW-LINE (Y/N)?
 - BKSP=00008/
 - CONSOLE LENGTH=00080/
 - HIGH MEMORY=16383/
 - LOWER CASE(Y/N)?
 - PAD=4/
 - RUBOUT=00127/
 - SAVE?
 -
9. Before executing SAVE, have the tape transport ready at the DUMP port.
10. To use TED-8 directly from the distribution tape, type the RETURN key at any time rather than a question prompt key. TED-8 responds

```
HEATH TED-8 # 3.03.00.
```

The Text Editor is ready to use.



Loading From Configured Tape

1. Load the tape in the tape transport.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. A single beep indicates a successful load.
5. Press GO on the H8 front panel.
6. Repeatedly press the space bar on your console.*
7. The console terminal responds with:

```
HEATH/WINTEK TEXT EDITOR #3.03.00.  
--
```

The Text Editor is ready to use in the configured form.

***NOTE:** You must type the spaces to allow TED-8 to determine the interface type and baud rate of your H8 Computer System. The number of spaces required depends on the type of console terminal, interface circuit board, and baud rate you are using.

APPENDIX B

Command Summary

Each of the commands for the Heath Text Editor are summarized in this Appendix. For a detailed explanation of each command, refer to “The Commands,” Page 3-14 and to Page 3-25 to “Terminal Control Characters,” Page 3-28.

Command Structure (Page 3-5)

The general form for an editor command is:

[<RANGE EXPRESSION>] [<VERB>] [<OPTION>] [<QUALIFIER STRING>] [<PARAMETERS>]

Range Expression Forms (Page 3-5)

1. NULL — First line in previous command range.
2. BLANK — The entire working buffer.
3. = — The previous command range.
4. A single line expression.
5. Two-line expressions.

Line Expression Forms (Pages 3-6 and 3-7)

	EXPRESSION	DEFINITION
1.	↑	The first line in the buffer.
2.	\$	The last line in the buffer.
3.	NULL	The first line in the previous command range.
4.	COMMA	Separates multiple line expressions.
5.	+COUNT	Move forward count decimal lines.
6.	-COUNT	Move backward count decimal lines.
7.	+‘STRING’	Move forward until ‘STRING’ is located.
8.	-‘STRING’	Move backward until ‘STRING’ is located.

Verb (Command) Forms

INSERT	Add to text buffer from keyboard. ESCAPE returns Editor to command mode (Page 3-14).
REPLACE	Replace eligible lines in command range from keyboard. ESCAPE returns Editor to command mode (Page 3-15).
DELETE	Deletes eligible lines in command range (*except entire buffer) (Page 3-16).
EDIT	Replaces old string with new string once per line. Parameter field: <arbitrary delimiter> old string <arbitrary delimiter> count. Count is decimal number of replacements (Page 3-17).
PRINT	Print eligible lines on console terminal (Page 3-18).
TAB	Sets TAB stops. Parameter field format is Tab 1, . . . , n. 1-n are up to 6 column numbers (Page 3-19).
BLITZ	Discards all text after a Y reply to SURE? (Page 3-20).
USE	Displays number of lines in buffer, bytes used, and bytes free (Page 3-20).
NEWIN	Opens a new tape file to be read in. Parameter field specifies file name <delimiter> <name> <delimiter> (Page 3-21).
FILL	Fills the working buffer with text from input tape. Stops at end-of-file or less than 512 free bytes (Page 3-22).
READ	Reads one additional block (512 bytes) from input tape (Page 3-22).

NEWOUT	Opens a new tape file for output. Parameter field specifies file name <delimiter> <name> <delimiter> (Page 3-22).
WRITE	Writes text from the working buffer to output tape. Writes start of buffer to first line of common range. After writing, lines are deleted (Page 3-23).
FLUSH	Writes working buffer, balance of input file, and end-of-file character onto output tape. Use when editing is complete. Text is deleted when complete. (Page 3-23).
NEXT	Writes working buffer onto output tape. Then fills buffer from input tape (Page 3-23).
SAVE	Saves text from the working buffer on the output tape. Saves start of buffer to first line of command range. Buffer is not deleted (Page 3-23).
SEARCH	Searches text buffer and input tape after initial line for 'STRING'. Search stops when STRING is found or end-of-file is found. Command range set to line containing 'STRING' (Page 3-24).
VERIFY	Verifies the contents of file 'name.' Performs a CRC and a sequence test on each record and compares this computed CRC to the recorded CRC (Page 3-25).
XINSERT	Add to the Text buffer from the XPORT. Text at the load port is uncompressed ASCII. ESCape or CTRL/C returns TED-8 to the command mode (Page 3-27).
STOP INPUT	Sets the EOF flag, indicating a closed file (Page 3-24).
XPORT	Specifies an alternate port for I/O operation (Page 3-25).

STOP OUTPUT

Writes an EOF record to tape. Closes current output file (Page 3-24).

XPRINT, nnn

Diverts printing to a line printer (Page 3-27).

Option (Page 3-11)

The option field is:

B Print line before operating on it.

A Print line after operating on it.

BA Print line before and after operating on it.

NULL No option specified.

Qualifier String (Page 3-12)

Qualifier string, if present, takes the form 'string'. A qualifier string limits range expression to those lines containing the qualifier string.

Parameter Field (Page 3-13)

This field contains extra parameters needed by the EDIT, TAB, NEWIN, and NEWOUT commands. Format is command dependent.

Terminal Control Characters (Page 3-28)

BACKSPACE (CONTROL H)	Deletes one character in command and text modes.
RUBOUT	Deletes line in command and text modes.
TAB (CONTROL-I)	Advances cursor to next TAB column.
CONTROL S	Sets suspend output flag.
CONTROL Q	Resets suspend output flag.
CONTROL O	Toggles discard output flag.
CONTROL P	Resets discard output flag.

APPENDIX C

The following edited source file is typical of one you might create using TED-8. It is intended to show examples of some of TED-8's editing capabilities. The assembly of this example is shown on Page 4-53.

add start
 -- PRINT
 FPNRM EQU 100000A-160Q
 INX B INC
 INX B TO
 INX B EXPONENT
 LDAX B
 ANA B SET CONDX CODE
 RZ
 DCR A /2
 JZ USRI IF UNDER FLOW
 DCR A /2 again (/4)
 USRI STAX B
 CALL FPNRM
 RET
 END START

change to 063207A
 073173A

--↑+'EQU' INSERTB
 FPNRM EQU 073173A
 START INX B INC
 --↑+'073173A' EDITBA,073173A,063207A,1
 FPNRM EQU 073173A
 FPNRM EQU 063207A
 -- PRINT
 ORG 100000A-160Q
 FPNRM EQU 063207A
 START INX B INC
 INX B TO
 INX B EXPONENT
 LDAX B
 ANA B SET CONDX CODE
 RZ
 DCR A /2
 JZ USRI IF UNDER FLOW
 DCR A /2 again (/4)
 USRI STAX B
 CALL FPNRM
 RET
 END START

add
 (A) = ACCX EXP
 Ret to ACCX
 Normalize

```

--↑+'EXPONENT'INSERTB
      INX      B      EXPONENT
      LDAX     A      (A)=ACCX EXP
--'ACC'+1DELETEDB
      LDAX     A
--↑+' /4'INSERTB
      DCR      A      /2AGAIN(/4)
      USRI     STAX   B      RET TO ACCX
--'ACCX'+1DELETEDB
      USRI     STAX   B
--'ACCX'INSERTB
      USRI     STAX   B      RET TO ACCX
      CALL     FPNRM  NORMALIZE
--'NORMALIZE'+1DELETEDB
      CALL     FPNRM
-- PRINT
      ORG      120000A-160Q
FPNRM  EQU      063207A
START  INX      B      INC UP
      INX      B      TO
      INX      B      EXPONENT
      LDAX     B      (A)=ACCX EXP
      ANA      A      SET CONDX CODE
      RZ
      DCR      A      /2
      JZ      USRI    IF UNDER FLOW
      DCR      A      /2AGAIN(/4)
USRI   STAX     B      RET TO ACCX
      CALL     FPNRM  NORMALIZE
      RET      IN CASE 0

      END      START

--NEWOUT"USR PROGRAM FOR BASIC #1.0"
--$SAVE
--STOP OUTPUT
SURE?Y
--VERIFY,5
RECORD #000 OK
RECORD #001 OK
RECORD #002 OK
*EOF*
--

```

INDEX

After A, 3-11
Arbitrary Delimiting Character, 3-18

Backspace, 3-28
Before (B), 3-11
Blank, 3-8
Blitz, 3-20
Buffer, 3-3

Checksum Error, 3-29
Command Completion, 3-30
Command Mode, 3-4
Command Structure, 3-5 ff.
Command Summary, 3-33
Compression (Blanks), 3-3
Control Characters, 3-28
Control C, 3-4, 3-15
Control I, 3-28
Control H, 3-28
CTRL-O, 3-29
CTRL-P, 3-29, 3-36
CTRL-Q, 3-29
CTRL-S, 3-29

Delete, 3-16

Equal (=), 3-10
Escape key, 3-4, 3-14, 3-15

Fill, 3-22
Flush, 3-23

Insert before first line, 3-14
Insert, 3-14

Loading TED-8, 3-31 ff,

Modes, 3-4
Multiple Line Expression, 3-7

Newin, 3-21
Newout, 3-22
Next, 3-23
Null, 3-9

Option Field, 3-11

Parameter Field, 3-13
Print, 3-18

Qualifier String, 3-12

Range Expressions, 3-6, 3-12
Read, 3-22
Replace, 3-14
Rubout, 3-28

Save, 3-23
Search, 3-24
Sequence Error, 3-29
Single-Line Expressions, 3-6
Stop, Input, 3-24
Stop, Output, 3-24
Sure, 3-20, 3-21

Tab, 3-19, 3-28
Tape Errors, 3-29
Text Mode, 3-4

Use, 3-20

Verb, 3-10
Verify, 3-25

Write, 3-23

XINSERT, 3-27
XPORT, 3-25
XPRINT, 3-27

SECTION 4

HEATH ASSEMBLY LANGUAGE

HASL-8



TABLE OF CONTENTS

WRITING H8 ASSEMBLY LANGUAGE PROGRAMS	4-3
THE CHARACTER SET	4-4
STATEMENTS	
The Label Field	4-5
The Opcode Field	4-5
The Operand Field	4-6
The Comment Field	4-6
Format Control	4-7
OPERAND EXPRESSIONS	
Operators	4-8
Tokens	4-8
THE 8080 OPCODES	4-10
Terms, Symbols, & Nomenclature	4-11
Data Transfer Group	4-17
Arithmetic Group	4-21
Logical Group:	4-28
Branch Group	4-34
Stack, I/O, and Machine Control Group	4-38
PSEUDO OPCODES/ASSEMBLER DIRECTIVES	
Define Byte, DB	4-44
Define Space, DS	4-44
Define Word, DW	4-45
Conditional Assembly Pseudo Operators	4-45
Listing Control	4-47
USING THE ASSEMBLER	
Errors	4-56
Control Characters	4-58
APPENDIX A	
Loading From the Software Distribution Tape	4-59
Loading From a Configured Tape	4-60
Index	4-61

WRITING H8 ASSEMBLY LANGUAGE PROGRAMS

The source code that the Heath Text Editor produces is translated into machine instructions by the Heath assembler, HASL-8. This source code consists of abbreviated English statements, numbers, and symbols that you will find much easier to understand and use than machine instructions. The translated source code that this assembler produces is called an object program. This object program may be executed directly from H8 memory or written to cassette tape. PAM-8 loads and executes a program that was written to tape.

This Manual assumes that you are familiar with the operation of the H8 microcomputer. This includes using PAM-8, a console, and one or two cassette recorders. Also, because of the many cross-references, we suggest that you read the complete manual on HASL-8 before using the assembler.

You should have a working knowledge of text editors, assemblers, input/output, and memory allocation to use assembly language programs efficiently. We suggest a course in assembly language programming, such as the Heath EC-1108 (to be announced in the Heathkit Catalog when available), to provide this knowledge if you are not familiar with using the 8080 microprocessor.

The assembler is capable of producing two different output programs. The object program is a binary image of the source code and contains the machine instructions that the 8080 microprocessor uses. The other program is a listing of the source code and the machine instructions. This listing is especially useful for program documentation and provides you with a method for detecting program errors.

HASL-8 is a two-pass assembler. This means that the listing and the object program are produced only after the assembler completes two separate passes through the source program. The first pass constructs a symbol table that assigns an address to each program label. During the second pass, the assembler translates the source program. It is this translation that produces the object and the listing program.

HASL-8 requires at least 8 kilobytes of random-access memory and a mass storage device. This device may consist of one or two independent cassette recorders. The assembler provides for approximately 250 user-defined symbols when used with 8K of memory.

THE CHARACTER SET

The Heath Assembly Language source program is composed of symbols, numbers, expressions, symbolic instructions, argument separators, assembly directives, and line terminators, all using ASCII characters. Those characters that are acceptable to HASL-8 are listed below.

1. The letters A through Z (lower case letters are acceptable for quoted strings and comments, provided that HASL-8 is configured properly in accordance with the software configuration guide).
2. The numerals 0 through 9.
3. The characters period (.) and dollar sign (\$), which are considered alphabetic.
4. The symbols

: = % # () , ; " ' + - _ ! ?

LINE FEED AND CARRIAGE RETURN

STATEMENTS

A source program is composed of a sequence of statements, designed to solve a problem. Each statement must be on a single line.

A statement is composed of up to four fields, identified by the order of appearance and by separating characters. The four fields are:

LABEL	OPCODE	OPERAND	COMMENT
-------	--------	---------	---------

The label and comment fields are optional. The opcode and operand fields are interdependent; either may be omitted, depending upon the contents of the other.

The Label Field

The label field always starts in column one. A label is a user-defined symbol assigned the current value of the memory location counter. It is a symbolic means of referring to a specific memory location within a program. Most statements do not require a label. If you do not want a label, column one must be left blank. Although the label is usually used to allow symbolic reference to the address of the labeled instruction, the SET and EQU pseudos make special use of the label field.

A label must start with an alphabetic character, and it consists entirely of alphabetic or numeric characters. The maximum length of a label is 7 characters. Note that the characters “\$” and “.” are considered alphabetic. Therefore, the following are valid labels.

```
A  A3  C9D4  .START  ..  $END  END$PGM
```

If the current location counter is set to 040 100 and the next statement is:

```
.START  MOV  A,B
```

the assembler assigns the value 040 100 to the label .START. Subsequent references to .START refer to location 040 100.

The Opcode Field

All statements (except the comment statements) must have an opcode field. The opcode field need not be located in any particular column. However, it must be separated from the label field by at least one blank. If no label is specified, the opcode field may start in or after column 2.



The opcode is either an instruction mnemonic or an assembler directive. When the entry in the opcode field is an instruction mnemonic, it specifies a machine operation to be performed on any following operands. When it is an assembler directive, it specifies certain functions or actions to be performed by the assembler during program assembly.

The opcode field is terminated by a blank or by the end of a line.

The Operand Field

The operand field follows the opcode field and must be separated from it by at least one blank. Not all opcodes require operands. The operand contains information used by a machine instruction or, in the case of assembler directives (pseudo opcodes or pseudo ops), it contains information to be used by the pseudo op.

Operands may be symbols, expressions, or numbers. When multiple operands appear with a statement, each is separated from the next by a comma. An operand may be followed by a comment.

The operand field is terminated by a blank when followed by a comment, or by the end of a line when the operand ends the assembly statement. For example,

```
.START    MOV A,B  THIS IS A COMMENT
```

The space between `.START` and `MOV` terminates the label field; the blank between `MOV` and `A,B` terminates the opcode field and begins the operand field. The comma separates the operands `A` and `B` and the blank terminates the operand field and begins the comment field.

The Comment Field

The comment field follows the operand field, or the opcode field if no operand field is present. It must be separated from its preceding field by at least one blank. The comment field is not processed by the assembler and it is designed to contain documentary information. The comment field is optional and may contain any printing ASCII character. All other characters, even those with special significance to the assembler, are ignored by the assembler when used in the comment field.

A statement with an asterisk (*) in column one is taken as a comment statement and is not otherwise processed by the assembler. A totally blank line is also taken as a comment.

Format Control

The format of an assembly language program is controlled by the blank character. Format control is primarily used to produce a program which is easily read. Format control has no affect on the assembly process of the source program, and because HASL-8 uses compression techniques, the use of multiple blanks does not take up extra memory space. The following two statements are identical with the exception that the first one does not use any tabs and the second one uses tabs.

```
.START MOV A,B  THIS IS A COMMENT  
.START      MOV A,B      THIS IS A COMMENT
```

The TABs were converted to the appropriate number of blanks by TED-8. Therefore, HASL-8 sees no TAB characters.



OPERAND EXPRESSIONS

Except when the opcode is a machine instruction requiring that an 8080 register be specified as the operand, all operand fields may be coded as operand expressions. Such operand expressions are made up of integers, symbols, a special origin symbol, and character strings which may be combined, using certain operators. The operand may also be the origin symbol. The expressions are said to be made up of operators and tokens. No parentheses are allowed nor is any operator precedence recognized. Therefore, evaluation is strictly left to right. The result of any expression must fall between $-32,767$ and $65,534$.

Operators

HASL-8 recognizes 5 operators. They are:

- $+$ Addition of an integer arithmetic expression.
- $-$ Subtraction of an integer arithmetic expression.
- $*$ Multiplication of an integer arithmetic expression.
- $/$ Division of an integer arithmetic expression.
- $-$ (unary) negation of a standard integer arithmetic expression.

Note, the unary minus is valid only as the first character in an expression. The following are examples of legitimate assembler operand expressions.

```
3+5
-2      (unary)
1+2*3
```

Note that the last example evaluates to 9 rather than 7, as the **assembler does not recognize any operator precedence**. Therefore it evaluates the expression from left to right.

Tokens

Heath Assembly Language recognizes four different tokens: integers, symbols, character strings, and the origin symbol. Each of these tokens has the limitations described in the following sections.

INTEGERS

Decimal integers ranging from 0 to 65,535 are allowed, but no decimal place may be specified. The radix of an integer expression is assumed to be decimal. However, you may specify binary, octal, offset octal, decimal, or hexadecimal. Specify them by using a post-radix symbol following the integer expression.

B	Binary
O or Q	Octal
D	Decimal
H	Hexadecimal
A	Offset Octal

For example:

<u>EXPRESSION</u>	<u>RADIX</u>	<u>DECIMAL VALUE</u>
00000011B	Binary	3
160Q	Octal (also 112O)	112
3200	Decimal (also 3200D)	3200
77000A	Offset octal	16128
021AH	Hexadecimal	282

<u>LEGAL INTEGER EXPRESSIONS</u>	<u>ILLEGAL INTEGER EXPRESSIONS</u>	<u>COMMENTS</u>
232	232.1	Decimals may not be specified
10111B	226B	Not a binary number
177Q	888	Not an octal number
A1FH	21C	No hex radix specified

If an integer expression evaluates to less than $-32,767$, or greater than $65,534$, an error code is flagged.

SYMBOLS

An expression may contain any user defined symbol. Although most symbols do not need to be defined sequentially before the referencing statement, some pseudo operators require all their operand symbols to be defined in earlier statements in the program. Such operators are said to require "pass one evaluation" and are documented in "The 8080 Opcodes" (Page 4-11). All symbols must consist of legal HASL-8 characters.

The # Symbol

If the pound sign (#) is the first character in an expression, the expression is evaluated as a 16-bit expression. After the expression is evaluated, the resultant value is masked to an 8-bit equivalent. Once this is done, a 16-bit operand may be



referenced in an instruction requiring 8 bits without causing an overflow (V) error. For example:

```
MVI    H, ADDR/256
MVI    L, #ADDR      (HL) = 16 bit address
```

In this example, the first line of code loads the H and L register pair (16-bit register) with the binary value associated with the label "ADDR" divided by 256. The second line of code immediately loads the L register (an 8-bit register) with the lower 8-bits of the binary value equated to the symbol ADDR in the symbol table. This process does not cause an overflow error, as the 16-bit binary equivalent of ADDR is masked to the least significant 8-bits before it is moved into the 8-bit L register.

CHARACTER STRING

A character string consisting of one or two legal characters may be used as a token in an HASL-8 expression. Such a character string is enclosed in a single quote (apostrophe). For example:

```
'A'      The character A (Value 101Q)
'GL'     The character string GL (Value 107 114A)
" "      The character quotation mark (Value 042Q)
```

THE ORIGIN SYMBOL, ORG

The current value of the origin counter may be referenced with the special symbol asterisk (*). NOTE: The assembler decides from the expression context whether the asterisk (*) represents the origin counter or is the multiplication operator. For example, the program

```
ORG     10
A EQU   ***
```

defines the symbol A to have the value 100. The first statement, ORG 10, sets the origin counter to the value 10. In the second statement, the label A is equated with the first asterisk, which the assembler presumes to be the symbol for the origin counter. This is multiplied by the third symbol, which the assembler also presumes to be the origin symbol. However, the middle asterisk is taken as the multiplication operator.

THE 8080 OPCODES*

Heath Assembly Language supports the standard 8080 machine opcodes. A review of the 8080 instruction set is presented on the following pages. Included in this review is a discussion of instruction and data formats, addressing modes, conditions flags, the symbols or abbreviations used in describing the 8080 instruction set, and the discussion of the format used to describe each instruction.

The 8080 instruction set includes five different types of instructions:

- **Data Transfer Group** — move data between registers or between memory and registers.
- **Arithmetic Group** — add, subtract, increment, or decrement data in registers or in memory.
- **Logical Group** — AND, OR, EXCLUSIVE-OR, compare, rotate, or complement data in registers or in memory.
- **Branch Group** — conditional and unconditional jump instructions, subroutine call instructions, and return instructions.
- **Stack, I/O and Machine Control Group** — includes I/O instructions, as well as instructions for maintaining the stack and internal control flags.

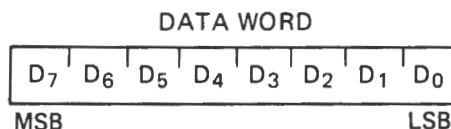
Terms, Symbols, & Nomenclature

INSTRUCTION AND DATA FORMATS

Memory for the 8080 is organized into 8-bit quantities called bytes. Each byte has a unique 16-bit binary address corresponding to its sequential position in memory.

The 8080 can directly address up to 65,536 bytes of memory, which may consist of both read-only memory (ROM) elements and random-access memory (RAM) elements (read/write memory).

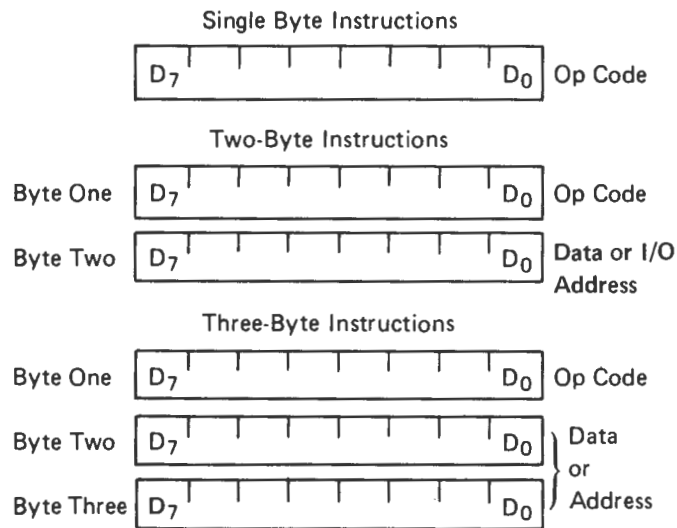
Data in the 8080 is stored in the form of 8-bit binary integers:



* Portions of this section are reprinted with the permission of Intel Corporation (Copyright, 1976).

When a register or data word contains a binary number, it is necessary to establish the order in which the bits of the number are written. In the Intel 8080, BIT 0 is referred to as the **Least Significant Bit (LSB)**, and BIT 7 (of an 8-bit number) is referred to as the **Most Significant Bit (MSB)**.

The 8080 program instructions may be one, two, or three bytes in length. Multiple byte instructions must be stored in successive memory locations; the address of the first byte is always used as the address of the instructions. The exact instruction format will depend on the particular operation to be executed.



ADDRESSING MODES

Often, the data that is to be operated on is stored in memory. When multi-byte numeric data is used, the data, like instructions, is stored in successive memory locations with the least significant byte first, followed by increasingly significant bytes. The 8080 has four different modes for addressing data stored in memory or in registers:

- **Direct** — Bytes 2 and 3 of the instruction contain the exact memory address of the data item (the low-order bits of the address are in byte 2, the high-order bits in byte 3).
- **Register** — Specifies the register or register pair in which the data is located.
- **Register Indirect** — Specifies a register pair which contains the memory address where the data is located (the high-order bits of the address are in the first register of the pair, the low-order bits in the second).

- **Immediate** — Contains the data itself. This is either an 8-bit quantity or a 16-bit quantity (least significant byte first, most significant byte second).

Unless directed by an interrupt or branch instruction, the execution of instructions proceeds through consecutively increasing memory locations. A branch instruction can specify the address of the next instruction to be executed in one of two ways:

- **Direct** — The branch instruction contains the address of the next instruction to be executed. (Except for the “RST” instruction, byte 2 contains the low-order address and byte 3 the high-order address.)
- **Register Indirect** — The branch instruction indicates a register pair which contains the address of the next instruction to be executed. (The high-order bits of the address are in the first register of the pair, the low-order bits in the second.)

The RST instruction is a special 1-byte call instruction (usually used during interrupt sequences). RST includes a 3-bit field; program control is transferred to the instruction whose address is eight times the contents of this 3-bit field.

CONDITION FLAGS

There are five condition flags associated with the execution of instructions on the 8080. They are Zero, Sign, Parity, Carry, and Auxiliary Carry, and are each represented by a 1-bit register in the CPU. A flag is “set” by forcing the bit to 1; and “reset” by forcing the bit to 0.

Unless indicated otherwise, when an instruction affects a flag, it affects it in the following manner.

- | | |
|--------|--|
| Zero: | If the result of an instruction has the value 0, this flag is set. Otherwise it is reset. |
| Sign: | If the most significant bit of the result of the operation has the value 1, this flag is set. Otherwise it is reset. |
| Parity | If the modulo 2 sum of the bits of the result of the operation is 0 (i. e., if the result has even parity), this flag is set. Otherwise it is reset (i. e., if the result has odd parity). |
| Carry: | If the instruction resulted in a carry (from addition), or a borrow (from subtraction or a comparison) out of the high-order bit, this flag is set. Otherwise it is reset. |

Auxiliary Carry: If the instruction caused a carry out of bit 3 and into bit 4 of the resulting value, the auxiliary carry is set. Otherwise it is reset. This flag is affected by single precision additions, subtractions, increments, decrements, comparisons, and logical operations, but is principally used with additions and increments preceding a DAA (Decimal Adjust Accumulator) instruction.

Symbols and Abbreviations

The following symbols and abbreviations are used in the subsequent description of the 8080 instructions:

SYMBOLS	MEANING
accumulator	Register A
addr	16-bit address quantity
data	8-bit data quantity
data 16	16-bit data quantity
byte 2	The second byte of the instruction
byte 3	The third byte of the instruction
port	8-bit address of an I/O device
r, r1, r2	One of the registers A,B,C,D,E,H,L
DDD, SSS	The bit pattern designating one of the registers A, B, C, D, E, H, L (DDD = destination, SSS = source):

DDD or SSS	REGISTER NAME
111	A
000	B
001	C
010	D
011	E
100	H
101	L

rp One of the register pairs:

B represents the B, C pair with B as the high-order register and C as the low-order register;

D represents the D, E pair with D as the high-order register and E as the low-order register;

H represents the H, L pair with H as the high-order register and L as the low-order register;

SP represents the 16-bit stack pointer register.

RP The bit pattern designating one of the register pairs B, D, H, SP:

RP	REGISTER PAIR
00	B-C
01	D-E
10	H-L
11	SP

rh The first (high-order) register of a designated register pair.

rl The second (low-order) register of a designated register pair.

PC 16-bit program counter register (PCH and PCL are used to refer to the high-order and low-order 8-bits respectively).

SP 16-bit stack pointer register (SPH and SPL are used to refer to the high-order and low-order 8-bits respectively).

rm Bit m of the register r (bits are numbered 7 through 0 from left to right).

Z, S, P, The condition flags:

Cy, AC

Zero,
Sign,
Parity,
Carry,
and Auxiliary Carry,
respectively.

NOTE: HASL-8 recognizes the E as well as the Z defining the zero bit. Therefore, JZ (jump zero) or JE (jump equal) are both valid op-codes.

() The contents of the memory location or registers enclosed in the parentheses.

← “Is transferred to”

\wedge Logical AND

∇ Exclusive OR

\vee Inclusive OR

+

 Addition

− Two’s complement subtraction

*

 Multiplication

↔ “Is exchanged with”

— The one’s complement (e. g., (A))

n The restart number 0 through 7

NNN The binary representation 000 through 111
for restart number 0 through 7 respectively.

Description Format

The following pages provide a detailed description of the instruction set of the 8080. Each instruction is described in the following manner:

1. The HASL-8 format, consisting of the opcode and operand fields, is printed in **BOLDFACE** on the left side of the first line.
2. The name of the instruction is enclosed in parentheses at the center of the first line.
3. The next line(s) contain a symbolic description of the operation of the instruction.

4. This is followed by a narrative description of the operation of the instruction.
5. The following line(s) contain the binary fields and patterns that comprise the machine instruction.
6. The last two lines contain incidental information about the execution of the instruction. The number of machine cycles and states required to execute the instruction are listed first. If the instruction has two possible execution times, as in a conditional jump, both times will be listed, separated by a slash. Next, any significant data addressing modes (see "Addressing Modes," Page 4-12) are listed. The last line lists any of the five Flags that are affected by the execution of the instruction.

Data Transfer Group

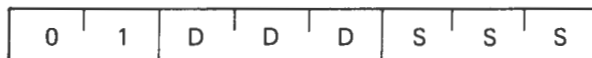
This group of instructions transfers data to and from registers and memory.

Condition flags are not affected by any instruction in this group.

MOV r1, r2 (Move Register)

$(r1) \leftarrow (r2)$

The content of register r2 is moved to register r1.



Cycles: 1

States: 5

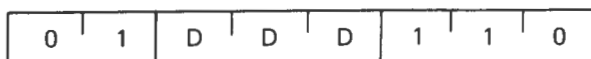
Addressing: register

Flags: none

MOV r, M (Move from memory)

$(r) \leftarrow ((H) (L))$

The content of the memory location whose address is in registers H and L is moved to register r.



Cycles: 2

States: 7

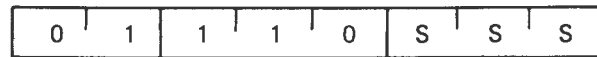
Addressing: reg. indirect

Flags: none

MOV M, r (Move to memory)

$((H) (L)) \leftarrow (r)$

The content of register r is moved to the memory location whose address is in registers H and L.



Cycles: 2

States: 7

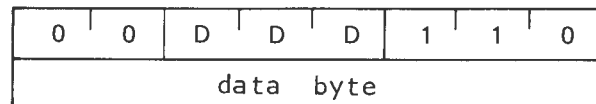
Addressing: reg. indirect

Flags: none

MVI r, data (Move to register immediate)

$(r) \leftarrow (\text{byte } 2)$

The content of byte 2 of the instruction is moved to register r.



Cycles: 2

States: 7

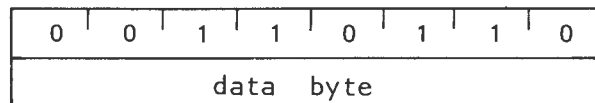
Addressing: immediate

Flags: none

MVI M, data (Move to memory immediate)

$((H) (L)) \leftarrow (\text{byte } 2)$

The content of byte 2 of the instruction is moved to the memory location whose address is in registers H and L.



Cycles: 3

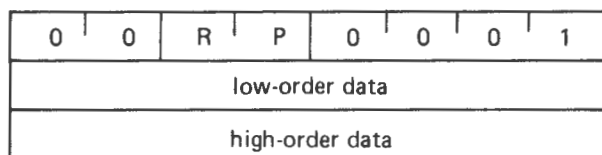
States: 10

Addressing: immed./reg.
indirect

Flags: none

LXI rp, data 16 (Load register pair immediate) $(rh) \leftarrow (\text{byte } 3),$ $(rl) \leftarrow (\text{byte } 2)$

Byte 3 of the instruction is moved into the high-order register (rh) of the register pair rp. Byte 2 of the instruction is moved into the low-order register (rl) of the register pair rp.



Cycles: 3

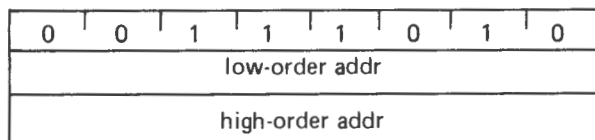
Addressing: immediate

States: 10

Flags: none

LDA addr (Load Accumulator direct) $(A) \leftarrow (\text{byte } 3) (\text{byte } 2)$

The content of the memory location, whose address is specified in byte 2 and byte 3 of the instruction, is moved to register A.



Cycles: 4

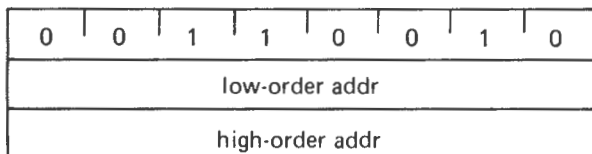
Addressing: direct

States: 13

Flags: none

STA addr (Store accumulator direct) $((\text{byte } 3) (\text{byte } 2)) \leftarrow (A)$

The content of the accumulator is moved to the memory location whose address is specified in byte 2 and byte 3 of the instruction.



Cycles: 4

Addressing: direct

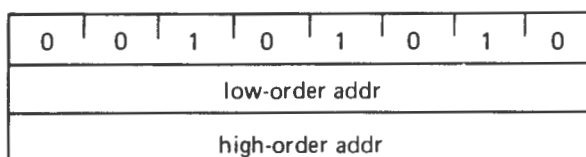
States: 13

Flags: none

LHLD addr (Load H and L direct)
$$(L) \leftarrow ((\text{byte } 3) (\text{byte } 2))$$

$$(H) \leftarrow ((\text{byte } 3) (\text{byte } 2) + 1)$$

The content of the memory location whose address is specified in byte 2 and byte 3 of the instruction is moved to register L. The content of the memory location at the succeeding address is moved to register H.



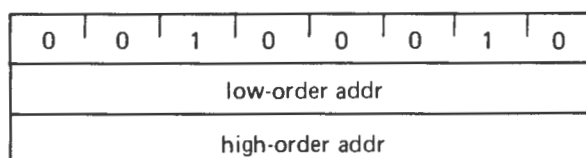
Cycles: 5
States: 16

Addressing: direct
Flags: none

SHLD addr (Store H and L direct)
$$((\text{byte } 3) (\text{byte } 2)) \leftarrow (L)$$

$$((\text{byte } 3) (\text{byte } 2) + 1) \leftarrow (H)$$

The content of register L is moved to the memory location whose address is specified in byte 2 and byte 3. The content of register H is moved to the succeeding memory location.



Cycles: 5
States: 16

Addressing: direct
Flags: none

LDAX rp (Load accumulator indirect)
$$(A) \leftarrow ((rp))$$

The content of the memory location whose address is in the register pair rp is moved to register A. NOTE: Only register pairs rp = B (registers B and C) or rp = D (registers D and E) may be specified.



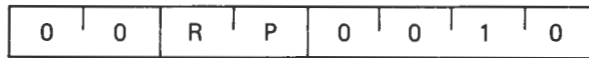
Cycles: 2
States: 7

Addressing: reg. indirect
Flags: none

STAX rp (Store accumulator indirect)

$((rp)) \leftarrow (A)$

The content of register A is moved to the memory location whose address is in the register pair rp. NOTE: Only register pairs rp = B (registers B and C) or rp = D (registers D and E) may be specified.



Cycles: 2

Addressing: reg. indirect

States: 7

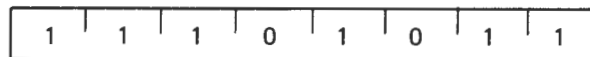
Flags: none

XCHG (Exchange H and L with D and E)

$(H) \longleftrightarrow (D)$

$(L) \longleftrightarrow (E)$

The contents of registers H and L are exchanged with the contents of registers D and E.



Cycles: 1

Addressing: register

States: 4

Flags: none

Arithmetic Group

This group of instructions performs arithmetic operations on data in registers and memory.

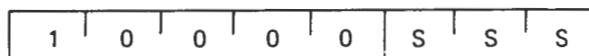
Unless indicated otherwise, all instructions in this group affect the Zero, Sign, Parity, Carry, and Auxiliary Carry flags according to the standard rules.

All subtraction operations are performed via two's complement arithmetic and set the carry flag to one to indicate a borrow and clear it to indicate no borrow.

ADD r (Add Register)

$(A) \leftarrow (A) + (r)$

The content of register r is added to the content of the accumulator. The result is placed in the accumulator.



Cycles: 1

Addressing: register

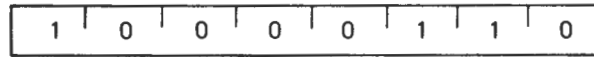
States: 4

Flags: Z,S,P,CY,AC

ADD M (Add memory)

$$(A) \leftarrow (A) + ((H) (L))$$

The content of the memory location whose address is contained in the H and L registers is added to the content of the accumulator. The result is placed in the accumulator.



Cycles: 2

Addressing: reg. indirect

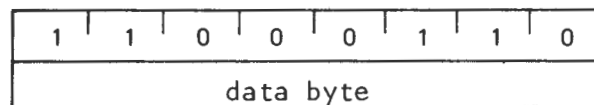
States: 7

Flags: Z,S,P,CY,AC

ADI DATA (add immediate)

$$(A) \leftarrow (A) + (\text{byte } 2)$$

The content of the second byte of the instruction is added to the content of the accumulator. The result is placed in the accumulator.



Cycles: 2

Addressing: immediate

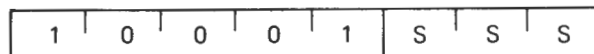
States: 7

Flags: Z,S,P,CY,AC

ADC r (Add Register with carry)

$$(A) \leftarrow (A) + (r) + (CY)$$

The content of register r and the content of the carry bit are added to the content of the accumulator. The result is placed in the accumulator.



Cycles: 1

Addressing: register

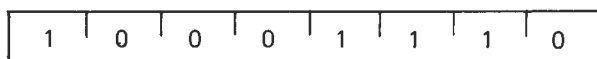
States: 4

Flags: Z,S,P,CY,AC

ADC M (Add memory with carry)

$$(A) \leftarrow (A) + ((H) (L)) + (CY)$$

The content of the memory location whose address is contained in the H and L registers and the content of the CY flag are added to the accumulator. The result is placed in the accumulator.



Cycles: 2

Addressing: reg. indirect

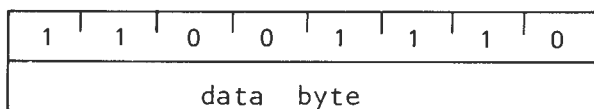
States: 7

Flags: Z,S,P,CY,AC

ACI data (Add immediate with carry)

$$(A) \leftarrow (A) + (\text{byte 2}) + (CY)$$

The content of the second byte of the instruction and the content of the CY flag are added to the contents of the accumulator. The result is placed in the accumulator.



Cycles: 2

Addressing: immediate

States: 7

Flags: Z,S,P,CY,AC

SUB r (Subtract Register)

$$(A) \leftarrow (A) - (r)$$

The content of register r is subtracted from the content of the accumulator. The result is placed in the accumulator.



Cycles: 1

Addressing: register

States: 4

Flags: Z,S,P,CY,AC

SUB M (Subtract memory)

$$(A) \leftarrow (A) - ((H) (L))$$

The content of the memory location whose address is contained in the H and L registers is subtracted from the content of the accumulator. The result is placed in the accumulator.



Cycles: 2

States: 7

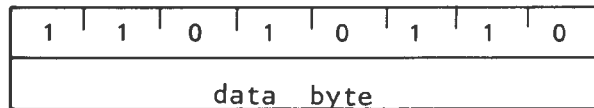
Addressing: reg. indirect

Flags: Z,S,P,CY,AC

SUI DATA (Subtract immediate)

$$(A) \leftarrow (A) - (\text{byte } 2)$$

The content of the second byte of the instruction is subtracted from the content of the accumulator. The result is placed in the accumulator.



Cycles: 2

States: 7

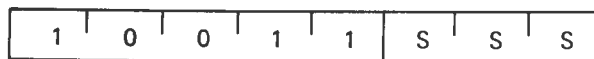
Addressing: immediate

Flags: Z,S,P,CY,AC

SBB r (Subtract Register with borrow)

$$(A) \leftarrow (A) - (r) - (CY)$$

The content of register r and the content of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.



Cycles: 1

States: 4

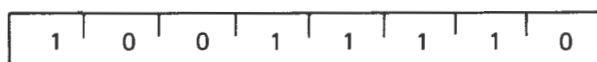
Addressing: register

Flags: Z,S,P,CY,AC

SBB M (Subtract memory with borrow)

$$(A) \leftarrow (A) - ((H) (L)) - (CY)$$

The content of the memory location whose address is contained in the H and I registers and the content of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.



Cycles: 2

Addressing: reg. indirect

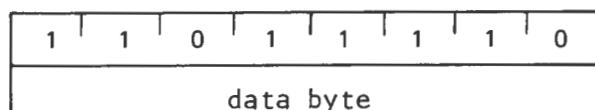
States: 7

Flags: Z,S,P,CY,AC

SBI data (Subtract immediate with borrow)

$$(A) \leftarrow (A) - (\text{byte 2}) - (CY)$$

The contents of the second byte of the instruction and the contents of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.



Cycles: 2

Addressing: immediate

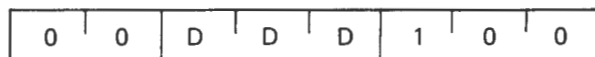
States: 7

Flags: Z,S,P,CY,AC

INR r (Increment Register)

$$(r) \leftarrow (r) + 1$$

The content of register r is incremented by one. NOTE: All condition flags **except** CY are affected.



Cycles: 1

Addressing: register

States: 5

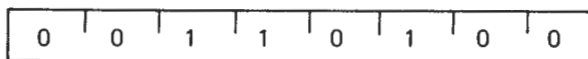
Flags: Z,S,P,AC



INR M (Increment memory)

$$((H) (L)) \leftarrow ((H) (L)) + 1$$

The content of the memory location whose address is contained in the H and L registers is incremented by one. NOTE: All condition flags **except CY** are affected.



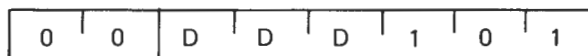
Cycles: 3
States: 10

Addressing: reg. indirect
Flags: Z,S,P,AC

DCR r (Decrement Register)

$$(r) \leftarrow (r) - 1$$

The content of register r is decremented by one. NOTE: All condition flags **except CY** are affected.



Cycles: 1
States: 5

Addressing: register
Flags: Z,S,P,AC

DCR M (Decrement memory)

$$((H) (L)) \leftarrow ((H) (L)) - 1$$

The content of the memory location whose address is contained in the H and L registers is decremented by one. NOTE: All condition flags **except CY** are affected.



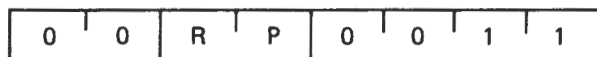
Cycles: 3
States: 10

Addressing: reg. indirect
Flags:

INX rp (Increment register pair) 0(0,2,4,6)3

$(rh) (rl) \leftarrow (rh) (rl) + 1$

The content of the register pair rp is incremented by one. NOTE: **No condition flags are affected.**



Cycles: 1

Addressing: register

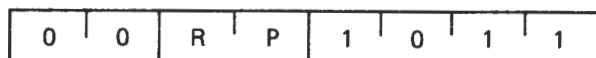
States: 5

Flags: none

DCX rp (Decrement register pair) 0(1,3,5,7)3

$(rh) (rl) \leftarrow (rh) (rl) - 1$

The content of register pair rp is decremented by one. NOTE: **No condition flags are affected.**



Cycles: 1

Addressing: register

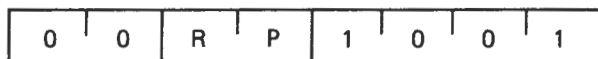
States: 5

Flags: none

DAD rp (Add register pair to H and L)

$(H) (L) \leftarrow (H) (L) + (rh) (rl)$

The content of register pair rp is added to the content of the register pair H and L. The result is placed in register pair H and L. NOTE: **Only the CY flag is affected.** It is set if there is a carry out of the double precision add; otherwise it is reset.



Cycles: 3

Addressing: register

States: 10

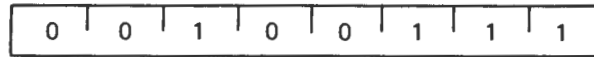
Flags: CY



DAA (Decimal Adjust Accumulator)

The eight-bit number in the accumulator is adjusted to form two 4-bit Binary-Coded-Decimal digits by the following process:

1. If the value of the least significant 4 bits of the accumulator is greater than 9 **or** if the AC flag is set, 6 is added to the accumulator.
2. If the value of the most significant 4 bits of the accumulator is now greater than 9, **or** if the CY flag is set, 6 is added to the most significant 4 bits of the accumulator.



Cycles:	1
States:	4
Flags:	Z,S,P,CY,AC

Logical Group:

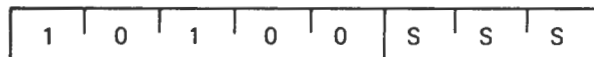
This group of instructions performs logical (Boolean) operations on data in registers and memory and on condition flags.

Unless indicated otherwise, all instructions in this group affect the Zero, Sign, Parity, Auxiliary Carry, and Carry flags according to the standard rules.

ANA r (AND Register)

$$(A) \leftarrow (A) \wedge (r)$$

The content of register r is logically anded with the content of the accumulator. The result is placed in the accumulator. **The CY flag is cleared.**

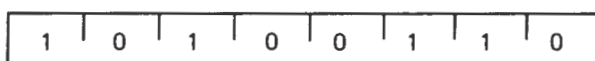


Cycles: 1	Addressing: register
States: 4	Flags: Z,S,P,CY,AC

ANA M (AND memory)

$$(A) \leftarrow (A) \wedge ((H) (L))$$

The contents of the memory location whose address is contained in the H and L registers is logically anded with the content of the accumulator. The result is placed in the accumulator. **The CY flag is cleared.**

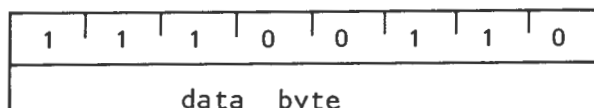


Cycles: 2 Addressing: reg. indirect
States: 7 Flags: Z,S,P,CY,AC

ANI data (AND immediate)

$$(A) \leftarrow (A) \wedge (\text{byte } 2)$$

The content of the second byte of the instruction is logically anded with the contents of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

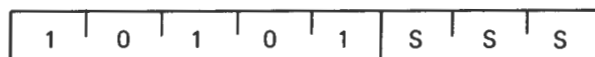


Cycles: 2 Addressing: immediate
States: 7 Flags: Z,S,P,CY,AC

XRA r (Exclusive OR Register)

$$(A) \leftarrow (A) \nabla (r)$$

The content of register r is exclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

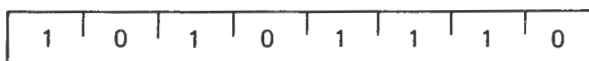


Cycles: 1 Addressing: register
States: 4 Flags: Z,S,P,CY,AC

XRA M (Exclusive OR Memory)

$$(A) \leftarrow (A) \nabla ((H) (L))$$

The content of the memory location whose address is contained in the H and L registers is exclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**



Cycles: 2

States: 7

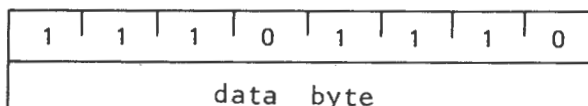
Addressing: reg. indirect

Flags: Z,S,P,CY,AC

XRI data (Exclusive OR immediate)

$$(A) \leftarrow (A) \nabla (\text{byte } 2)$$

The content of the second byte of the instruction is exclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**



Cycles: 2

States: 7

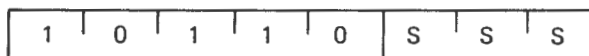
Addressing: immediate

Flags: Z,S,P,CY,AC

ORA r (OR Register)

$$(A) \leftarrow (A) \vee (r)$$

The content of register r is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**



Cycles: 1

States: 4

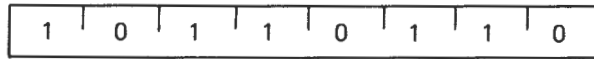
Addressing: register

Flags: Z,S,P,CY,AC

ORA M (OR memory)

$$(A) \leftarrow (A) \vee ((H) (L))$$

The content of the memory location whose address is contained in the H and L registers is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**



Cycles: 2

Addressing: reg. indirect

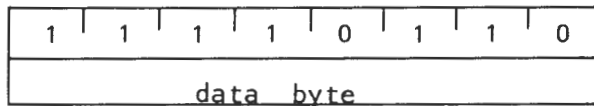
States: 7

Flags: Z,S,P,CY,AC

ORI data (OR Immediate)

$$(A) \leftarrow (A) \vee (\text{byte } 2)$$

The content of the second byte of the instruction is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**



Cycles: 2

Addressing: immediate

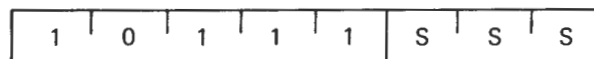
States: 7

Flags: Z,S,P,CY,AC

CMP r (Compare Register)

$$(A) - (r)$$

The content of register r is subtracted from the accumulator. The accumulator remains unchanged. The condition flags are set as a result of the subtraction. **The Z flag is set to 1 if $(A) = (r)$. The CY flag is set to 1 if $(A) < (r)$.**



Cycles: 1

Addressing: register

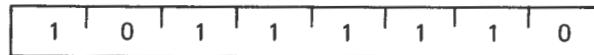
States: 4

Flags: Z,S,P,CY,AC

CMP M (Compare memory)

(A) — ((H) (L))

The content of the memory location whose address is contained in the H and L registers is subtracted from the accumulator. The accumulator remains unchanged. The condition flags are set as a result of the subtraction. The Z flag is set to 1 if (A) = ((H) (L)). The CY flag is set to 1 if (A) < ((H) (L)).



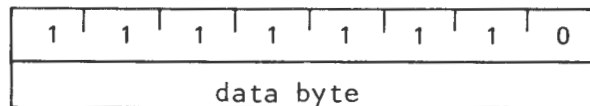
Cycles: 2
States: 7

Addressing: reg. indirect
Flags: Z,S,P,CY,AC

CPI data (Compare immediate)

(A) — (byte 2)

The content of the second byte of the instruction is subtracted from the accumulator. The condition flags are set by the result of the subtraction. The Z flag is set to 1 if (A) = (byte 2). The CY flag is set to 1 if (A) < (byte 2).



Cycles: 2
States: 7

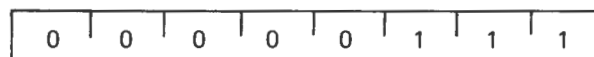
Addressing: immediate
Flags: Z,S,P,CY,AC

RLC (Rotate left)

$(A_n) \leftarrow (A_{n-1}); (A_0) \leftarrow (A_7)$

$(Cy) \leftarrow (A_7)$

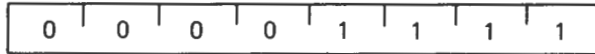
The content of the accumulator is rotated left one position. The low order bit and the CY flag are both set to the value shifted out of the high order bit position. **Only the CY flag is affected.**



Cycles: 1
States: 4
Flags: CY

RRC (Rotate right) $(A_n) \leftarrow (A_{n+1}); (A_7) \leftarrow (A_0)$ $(CY) \leftarrow (A_0)$

The content of the accumulator is rotated right one position. The high order bit and the CY flag are both set to the value shifted out of the low order bit position. **Only the CY flag is affected.**



Cycles: 1

States: 4

Flags: CY

RAL (Rotate left through carry) $(A_n) \leftarrow (A_{n-1}); (CY) \leftarrow (A_7)$ $(A_0) \leftarrow (CY)$

The content of the accumulator is rotated left one position through the CY flag. The low order bit is set equal to the CY flag and the CY flag is set to the value shifted out of the high order bit. **Only the CY flag is affected.**



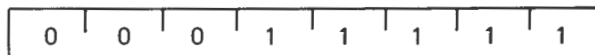
Cycles: 1

States: 4

Flags: CY

RAR (Rotate right through carry) $(A_n) \leftarrow (A_{n-1}); (CY) \leftarrow (A_0)$ $(A_7) \leftarrow (CY)$

The content of the accumulator is rotated right one position through the CY flag. The high order bit is set to the CY flag and the CY flag is set to the value shifted out of the low order bit. **Only the CY flag is affected.**



Cycles: 1

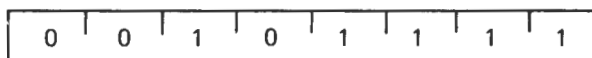
States: 4

Flags: CY

CMA (Complement accumulator)

$(A) \leftarrow (\overline{A})$

The contents of the accumulator are complemented (zero bits become 1, one bits become 0). **No flags are affected.**

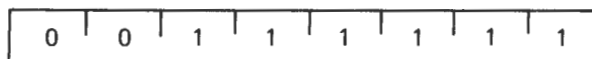


Cycles: 1
States: 4
Flags: none

CMC (Complement carry)

$(CY) \leftarrow (\overline{CY})$

The CY flag is complemented. **No other flags are affected.**

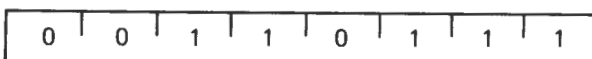


Cycles: 1
States: 4
Flags: CY

STC (Set carry)

$(CY) \leftarrow 1$

The CY flag is set to 1. **No other flags are affected.**



Cycles: 1
States: 4
Flags: CY

Branch Group

This group of instructions alter normal sequential program flow. **Condition flags are not affected** by any instruction in this group.

The two types of branch instructions are unconditional and conditional. Unconditional transfers simply perform the specified operation on register PC (the

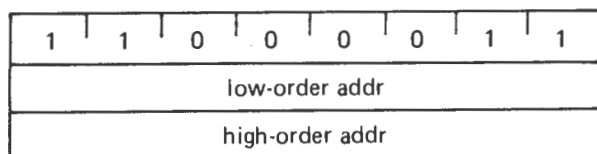
program counter). Conditional transfers examine the status of one of the four processor flags to determine if the specified branch is to be executed. The following conditions may be specified:

CONDITION	CCC	OCTAL
NE or NZ — not zero ($Z=0$)	000	0
E or Z — zero ($Z=1$)	001	1
NC — no carry ($CY = 0$)	010	2
C — carry ($CY = 1$)	011	3
PO — parity odd ($P = 0$)	100	4
PE — parity even ($P = 1$)	101	5
P — plus ($S = 0$)	110	6
M — minus ($S = 1$)	111	7

JMP addr (Jump)

$(PC) \leftarrow (\text{byte } 3) (\text{byte } 2)$

Control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction.



Cycles: 3

States: 10

Addressing: immediate

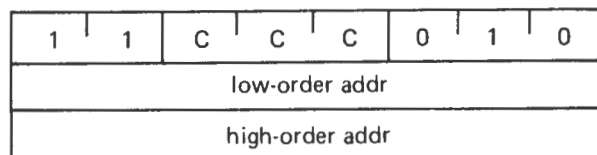
Flags: none

JNE JNC JPO JP
JE JC JPE JM (Condition jump)

If (CCC),

$(PC) \leftarrow (\text{byte } 3) (\text{byte } 2)$

If the specified condition is true, control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction. Otherwise, control continues sequentially.



Cycles: 3

States: 10

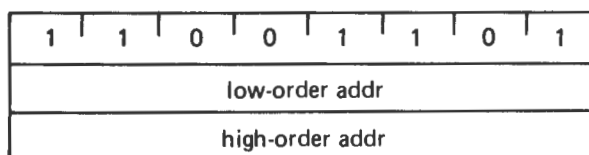
Addressing: immediate

Flags: none

CALL addr (Call)

$((SP) - 1) \leftarrow (PCH)$
 $((SP) - 2) \leftarrow (PCL)$
 $(SP) \leftarrow (SP) - 2$
 $(PC) \leftarrow (\text{byte 3}) (\text{byte 2})$

The high-order eight bits of the next instruction address are moved to the memory location whose address is one less than the content of register SP. The low-order eight bits of the next instruction address are moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by 2. Control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction.



Cycles: 5

Addressing: immediate/reg.
indirect

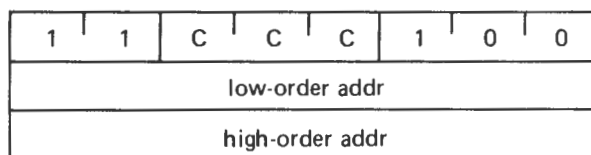
States: 17

Flags: none

CNE	CNC	CPO	CP	(Condition call)
CE	CC	CPE	CM	

If (CCC),
 $((SP) - 1) \leftarrow (PCH)$
 $((SP) - 2) \leftarrow (PCL)$
 $(SP) \leftarrow (SP) - 2$
 $(PC) \leftarrow (\text{byte 3}) (\text{byte 2})$

If the specified condition is true, the actions specified in the CALL instruction (see above) are performed; otherwise, control continues sequentially.



Cycles: 3/5

Addressing: immediate/reg.
indirect

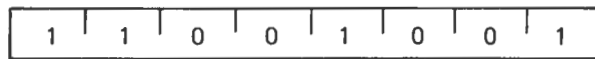
States: 11/17

Flags: none

RET (Return)

$(PCL) \leftarrow ((SP))$;
 $(PCH) \leftarrow ((SP)) + 1$;
 $(SP) \leftarrow (SP) + 2$;

The content of the memory location whose address is specified in register SP is moved to the low-order eight bits of register PC. The content of the memory location whose address is one more than the content of register SP is moved to the high-order eight bits of register PC. The content of register SP is incremented by 2.

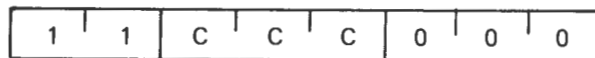


Cycles: 3 Addressing: reg. indirect
 States: 10 Flags: none

RNE	RNC	COP	CP	(Conditional return)
RE	RC	CPE	CM	

If (CCC),
 $(PCL) \leftarrow ((SP))$
 $(PCH) \leftarrow ((SP) + 1)$
 $(SP) \leftarrow (SP) + 2$

If the specified condition is true, the actions specified in the RET instruction (see above) are performed: otherwise, control continues sequentially.



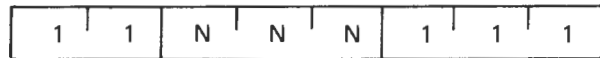
Cycles: 1/3 Addressing: reg. indirect
 States: 5/11 Flags: none

RST n (Restart)

$((SP) - 1) \leftarrow (PCH)$
 $((SP) - 2) \leftarrow (PCL)$
 $(SP) \leftarrow (SP) - 2$
 $(PC) \leftarrow 8 * (NNN)$

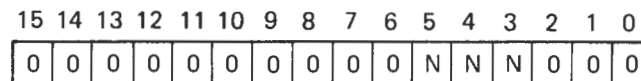
The high-order eight bits of the next instruction address are moved to the memory location whose address is one less than the content of register SP. The low-order eight bits of the next instruction address are moved to the memory location whose address is two less than the content of register SP.

The content of register SP is decremented by two. Control is transferred to the instruction whose address is eight times the content of NNN.



Cycles: 3
States: 11

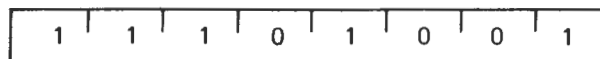
Addressing: reg. indirect
Flags: none



Program Counter After Restart

PCHL (Jump H and L indirect — move H and L to PC)
 $(PCH) \leftarrow (H)$
 $(PCL) \leftarrow (L)$

The content of register H is moved to the high-order eight bits of register PC.
 The content of register L is moved to the low-order eight bits of register PC.



Cycles: 1
States: 5

Addressing: register
Flags: none

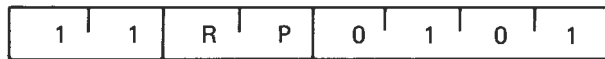
Stack, I/O, and Machine Control Group

This group of instructions performs I/O, manipulates the Stack, and alters internal control flags. Unless otherwise specified, **condition flags are not affected by any instructions in this group.**

PUSH rp (Push)

$((SP) - 1) \leftarrow (rh)$
 $((SP) - 2) \leftarrow (rl)$
 $(SP) \leftarrow (SP) - 2$

The content of the high-order register of register pair *rp* is moved to the memory location whose address is one less than the content of register *SP*. The content of the low-order register of register pair *rp* is moved to the memory location whose address is two less than the content of register *SP*. The content of register *SP* is decremented by 2. **NOTE: Register pair *rp* = *SP* may not be specified.**



Cycles: 3

Addressing: reg. indirect

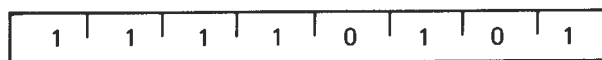
States: 11

Flags: none

PUSH PSW (Push processor status word)

$((SP) - 1) \leftarrow (A)$
 $((SP) - 2)_0 \leftarrow (CY), ((SP) - 2)_1 \leftarrow 1$
 $((SP) - 2)_2 \leftarrow (P), ((SP) - 2)_3 \leftarrow 0$
 $((SP) - 2)_4 \leftarrow (AC), ((SP) - 2)_5 \leftarrow 0$
 $((SP) - 2)_6 \leftarrow (Z), ((SP) - 2)_7 \leftarrow (S)$
 $(SP) \leftarrow (SP) - 2$

The content of register *A* is moved to the memory location whose address is one less than register *SP*. The contents of the condition flags are assembled into a processor status word and the word is moved to the memory location whose address is two less than the content of register *SP*. The content of register *SP* is decremented by two.



Cycles: 3

Addressing: reg. indirect

States: 11

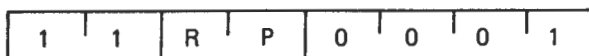
Flags: none

FLAG WORD

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
S	Z	0	AC	0	P	1	CY

POP rp (Pop)
$$\begin{aligned} (rl) &\leftarrow ((SP)) \\ (rh) &\leftarrow ((SP) + 1) \\ (SP) &\leftarrow (SP) + 2 \end{aligned}$$

The content of the memory location whose address is specified by the content of register SP is moved to the low-order register of register pair rp. The content of the memory location whose address is one more than the content of register SP is moved to the high-order register of register pair rp. The content of register SP is incremented by 2. **NOTE: Register pair rp = SP may not be specified.**

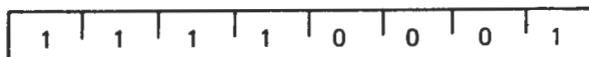


Cycles: 3
States: 10

Addressing: reg. indirect
Flags: none

POP PSW (Pop processor status word)
$$\begin{aligned} (CY) &\leftarrow ((SP))_0 \\ (P) &\leftarrow ((SP))_2 \\ (AC) &\leftarrow ((SP))_4 \\ (Z) &\leftarrow ((SP))_6 \\ (S) &\leftarrow ((SP))_7 \\ (A) &\leftarrow ((SP) + 1) \\ (SP) &\leftarrow (SP) + 2 \end{aligned}$$

The content of the memory location whose address is specified by the content of register SP is used to restore the condition flags. The content of the memory location whose address is one more than the content of register SP is moved to register A. The content of register SP is incremented by 2.



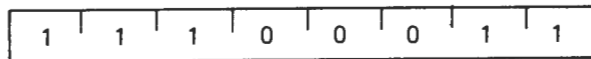
Cycles: 3
States: 10

Addressing: reg. indirect
Flags: Z,S,P,CY,AC

XTHL (Exchange stack top with H and L)

$(L) \longleftrightarrow ((SP))$
 $(H) \longleftrightarrow ((SP) + 1)$

The content of the L register is exchanged with the content of the memory location whose address is specified by the content of register SP. The content of the H register is exchanged with the content of the memory location whose address is one more than the content of register SP.

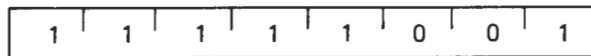


Cycles: 5 Addressing: reg. indirect
 States: 18 Flags: none

SPHL (Move HL to SP)

$(SP) \leftarrow (H) (L)$

The contents of registers H and L (16 bits) are moved to register SP.

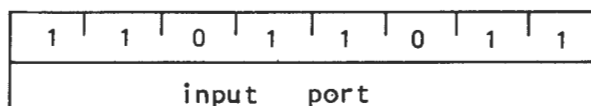


Cycles: 1 Addressing: register
 States: 5 Flags: none

IN port (Input)

$(A) \leftarrow (\text{data})$

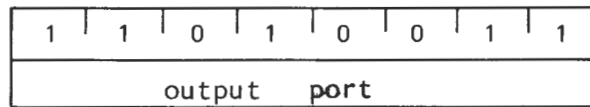
The data placed on the eight bit bidirectional data bus by the specified port is moved to register A.



Cycles: 3 Addressing: direct
 States: 10 Flags: none

OUT port (Output)(data) \leftarrow (A)

The content of register A is placed on the eight bit bidirectional data bus for transmission to the specified port.



Cycles: 3

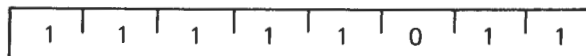
Addressing: direct

States: 10

Flags: none

EI (Enable interrupt)

The interrupt system is enabled **following the execution of the next instruction.**



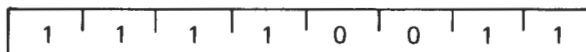
Cycles: 1

States: 4

Flags: none

DI (Disable interrupt)

The interrupt system is disabled **immediately following the execution of the DI instruction.**



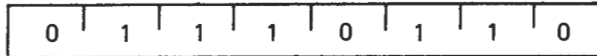
Cycles: 1

States: 4

Flags: none

HLT (Halt)

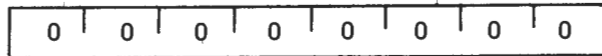
The processor is stopped. The registers and flags are unaffected.



Cycles: 1
States: 7
Flags: none

NOP (No op)

No operation is performed. The registers and flags are unaffected.



Cycles: 1
States: 4
Flags: none

PSEUDO OPCODES/ASSEMBLER DIRECTIVES

The Heath Assembly Language supports 20 assembler directives or, as they are more commonly known, pseudo opcodes or simply pseudo ops. These opcodes are called “pseudo” because they are coded as machine operations. But as their alternate name (assembler directives) indicates, they represent commands to HASL-8 and are not translated as instructions for the H8. Some pseudo ops conditionally affect the operation of the assembler. Others cause the assembler to generate constants into the generated object code.

Define Byte, DB

The DB pseudo defines byte contents. The DB pseudo is of the form:

```
Label DB iexp1, . . . . . ,iexpn
```

The integer expressions iexp1 through iexpn are expressions which evaluate to 8-bit values. For the DB pseudo, a long string can be substituted for an expression. The long string is a character string, delimited by single quotes ('), containing one or more characters. You can enclose a quote (') within a string by coding it as two single quotes. Each of the expressions is converted into an 8-bit binary number and stored in sequential memory locations. A few examples of the DB pseudo are:

CR	EQU	15Q
LF	EQU	12Q
	DB	1
	DB	2,3,4
	DB	10,CR,LF,'H8 BASIC',0

In each case, the DB pseudo converts the expression into a single byte and stores it in the appropriate memory location. The DB pseudo recognized a character string as a series of expressions. Therefore, each character is converted into its ASCII binary equivalent and is stored in a sequential memory location.

Define Space, DS

The defined space pseudo (DS) reserves a block of memory during assembly.

The form of the DS pseudo is:

```
LABEL DS iexp COMMENT
```

This pseudo is used, for example, to set up a buffer area or to define any other storage area. The DS pseudo causes the assembler to reserve a number of bytes specified by the expression (iexp) in the operand. These bytes are not preset to any value. Therefore, you should not presume any special original contents.

Programs using extensive buffer area should use the DS pseudo to declare this area. Using the DS pseudo significantly shortens the program load time. In the example

```
LINE    DS  80    80 character input line buffer
```

an 80-character input buffer is reserved by a single statement.

Define Word, DW

The DW pseudo defines word constants. The form of the DW pseudo is:

```
LABEL    DW    iexpl , . . . . . , iexpn
```

The DW pseudo specifies one or more data words iexp through iexpn. Data words are **2-byte** values which are placed into memory space, low order byte first. NOTE: Strings greater than two characters long are not allowed when you are using the DW pseudo.

Conditional Assembly Pseudo Operators

Frequently, you may want to write a program with certain portions of it that can be turned on or turned off. That is to say, when they are turned on, these portions of the program are assembled. If they are turned off, they are not assembled during that particular assembly. HASL-8 contains three pseudos to aid in conditional assembly. They are:

IF ELSE and ENDIF

IF

The IF pseudo conditionally disables assembly of any statements following the IF pseudo operator. The form of the IF pseudo operator is:

```
IF      iexp
```

If the expression (iexp) evaluates to zero, the statements following the IF pseudo are assembled. If the expression does not evaluate to zero (either negative or positive), any statements in the assembly source code following this expression are skipped until one of the three following pseudos are encountered. The ELSE, ENDIF and END pseudos are not skipped regardless of the value of the expression "iexp".

ELSE

The ELSE pseudo toggles the state of the assembly conditions. The ELSE pseudo

is of the form:

ELSE

If the conditional assembly flag is set to skip assembling source code, it is changed so source code is now assembled. If lines of source code prior to encountering the ELSE pseudo are being assembled, those following the ELSE pseudo are skipped until an ELSE, ENDIF, or END is encountered. NOTE: The ELSE segment must appear after an IF statement, but before the associated ENDIF statement.

ENDIF

The ENDIF statement indicates the end of a block of source code designated for conditional assembly. The form of the ENDIF pseudo is:

ENDIF

Assembly resumes regardless of the current assembly state (assembling or skipping) when the ENDIF conditional assembly pseudo occurs.

END

The END pseudo indicates the END of a program. The END pseudo takes the form:

END iexp

where iexp is the program entry point. The program entry point is the memory address where program execution begins. If the END statement is missing, the assembler generates one and prints an error message. If iexp is missing, the H8 does not receive a starting value for the program counter where the binary tape is loaded.

EQU

The Equate statement is used to assign an arbitrary value to a symbol. The form of the equate statement is:

LABEL EQU iexp

The equate statement is unique, as it must evaluate on pass one. For this reason, any symbols used within the expression “iexp” must be defined before the assembler encounters the EQU statements. The label is assigned the value of the integer expression “iexp”. This label may not be redefined by subsequent use as a label in any other statement. For example,

.START EQU *

The label .START is set equal to the value of the memory location counter, or

```
.START EQU 100
```

The label .START is set equal to 100.

NOTE: If you omit the label, an error is generated.

ORG

The Origin statement (ORG) sets the initial value of the memory location counter. The form of the origin statement is:

```
LABEL ORG iexp
```

The expression iexp must evaluate on pass one. Therefore, any symbols used within this expression must be defined before the assembler encounters this statement. When the assembler encounters the ORG statement, the memory location counter is set to the expression value. All subsequent object code generated by the assembler is placed in sequential memory locations, starting at the address given by the expression. It is legal to establish a new origin, either before or after a previous origin. If a label is present, it is given the value iexp. For example:

```
BEGIN ORG 40 100A
```

The program is started at location 040 100 (offset octal) and the label BEGIN is assigned the offset octal value 040 100.

```
BEGIN ORG .START+256
```

The memory location counter is set to the previously defined value of the label .START +256. The label BEGIN also assumes this value.

SET

The SET statement assigns an arbitrary value to a desired symbol. The form of the SET statement is:

```
LABEL SET iexp
```

The SET pseudo op differs from the EQU pseudo op in that any label defined in a SET statement can be redefined in a following SET statement as many times as desired in the course of the program. The expression “iexp” must evaluate during pass one. Therefore, any symbols used within the expression “iexp” must be previously defined.

Listing Control

HASL-8 provides a number of pseudo operators which affect the listing mode. They control paging, pagination, titles, and subtitles. The listing control pseudos are used to affect easily read documentation; they do not appear in the program listing.



TITLE

The pseudo operator TITLE causes a new page title to be used. The form of the title pseudo op is:

```
TITLE    'new title'
```

Unless the assembler is already at the top of a page, a new page of the assembly listing is generated. This page is given the title contained in the string 'new title'.

STL

The subtitle pseudo (STL) causes a new page subtitle to be set. The form of the subtitle pseudo is:

```
STL      'new subtitle'
```

The subtitle pseudo does not affect pagination. This is to say, it does not generate a new page but simply titles a subsection of the program. Subtitles are frequently used to indicate subroutines or major program modules.

EJECT

The EJECT pseudo causes a new page to be started. The form of the eject pseudo is:

```
EJECT
```

When HASL-8 processes an EJECT pseudo, the output device is instructed to move to the start of a new page during the listing.

SPACE

The space pseudo leaves blank lines in the program listing. The form of the space pseudo is:

```
SPACE    iexp1,iexp2
```

During the assembly listing, iexp1 blank lines are left. If the optional expression iexp2 is specified, the assembler checks during a listing to see if the number of lines remaining on the page is greater than or less than iexp2. If there are less than iexp2 lines remaining on the page, the spacing function is skipped and a new page is started, as if an EJECT pseudo was executed.

LON (Listing on)

The LON pseudo operator is used to turn-on listing options. The form of the LON pseudo is:

```
LON   CCC
```

Each option is represented by a single character. The characters for the desired options are supplied as CCC. The options and their default modes (if they are not specified) are:

L Master listing

If this option is enabled, all program lines are listed. If it is disabled, only lines containing errors are listed.

DEFAULT MODE: All program lines are listed (normally enabled; disable using LOF).

I Lists the IF-skipped lines. When this option is enabled, all lines skipped due to IF statements are listed (although they are not assembled).

DEFAULT MODE: The skip lines are not contained in the listing.

G Lists all generated bytes. When this option is enabled, all generated bytes appear on the listing. If more than three bytes are generated by a statement, new lines are generated in the listing to display these bytes. NOTE: Define byte pseudo can produce many bytes when you are encoding a string. These are not normally listed.

DEFAULT MODE: Lists a maximum of the 3-bytes generated in each statement.

LOF (Listing off)

The LOF pseudo is identical to the LON pseudo except that the selected options are disabled. The form of the LOF pseudo is:

```
LOF   CCC
```

See LON, above, for a description of the control character CCC.

ERRxx

HASL-8 contains four conditional error pseudo operators. These are of the form:

```
ERRZR   iexp  
ERRNZ   iexp  
ERRPL   iexp  
ERRMI   iexp
```

For each of these pseudo operators, the assembler tests the indicated expression. If the expression matches the expressed error condition, an error code is flagged in the listing. The errors associated with each of the conditional error pseudos are:

ERRZR	tests for zero expression
ERRNZ	tests for non-zero expression
ERRPL	tests for positive expression
ERRMI	tests for negative expression

These pseudo error tests are particularly useful when you make assumptions about the configuration of various program elements or expressions. You can encode these assumptions into ERRxx pseudos. So any change which causes the code to fail generates an error, flagging the programmer during the listing. For example,

LXI	H, AREA1	
MOV	B, M	(B) = (AREA1)
INX	H	
ERRNZ	AREA2-Area1-1	Assume area 2 follows area 1
MOV	C, M	(C) = (AREA2)

If, when the program is assembled, AREA 1 and AREA 2 have been defined differently, an error flag would warn of this mistake.

USING THE ASSEMBLER

Before you can use the Heath Assembly Language, you must prepare the source program using a Text Editor such as TED-8. Once the source program is prepared and stored on tape, you can load Heath Assembly Language in the H8. The loading procedure is outlined in "Appendix A" (Page 4-59). Once a configured version of HASL-8 is loaded and started, a series of questions must be answered. First the assembler asks

```
LISTING TO PRINTER (Y/N) <N> ?
```

in order to find out where the program listing should be printed. The "(Y/N)" tells you that the acceptable replies to the question are the letters "Y" and "N". The '<N>' tells you that the default reply is "N". That is, if you just type the RETURN key in reply to the question, HASL will assume you meant "N". (The text printed below in italics applies only when you type "Y".)

If you wish the output listing to appear on your console terminal, type N or just the RETURN key. If you have some other output device, perhaps an H14 printer or an H36 terminal, reply 'Y' to this question. If you reply 'N', HASL skips the following questions, and resumes questioning with 'BINARY TAPE?'. If you reply 'Y', HASL then asks

```
LISTING PORT <340Q>?
```

HASL-8 is asking for the port number to which you connected your printer device. The '<340Q>' means that if you just reply by pressing the RETURN key, HASL-8 will assume you meant port 340Q. The 'Q' is the numbers 'postradix', a way of telling HASL-8 that the number is in base-8 (octal). Valid postradixes are:

	Base 10 (decimal - - blank)
Q	Base 8 (octal - - Q)
B	Base 2 (binary - - B)

After you specify the printer port, HASL will ask:

```
H14 PRINTER (Y/N) <Y> ?
```

Once again, the '(Y/N)' indicates the legal replies to the question. The default reply is enclosed in the angle-brackets. If you specified port 340Q, the system printer port, in reply to the earlier question, the default reply will be 'Y'. If you specified some other port, the default reply will be 'N'. HASL next asks,

```
PAGE SIZE <60>?
```



to determine how many lines to print on each page. The default of 60 is the best number for standard 66-line computer forms. Note that you should enter the number of lines HASL-8 is to print on each page, not how big the pages are. For example a normal computer form (11 inches deep) has room for 66 lines, but you will want to respond to this question with 60 to allow a 6 line gap between pages to skip the perforations in the forms.

At this point, HASL has automatically determined the interface type at the port you specified. If you reply "Y" to the H14 question, HASL then 'knows' the proper baud rate and page sizes to use, and will proceed to the 'BINARY?' question. If you reply "N", HASL will question you further:

INTER-PAGE GAP SIZE <6>?

Since you are not using an H14 printer, which automatically handles inter-page spacing, you must tell HASL how many blank lines to leave between the bottom of one page and the top of the next. This number should be large enough to skip the perforations in fan-fold paper, or to leave a visible gap in roll paper. The default value of 6 is the best value to use with standard 66-line computer forms. The sum of the two replies to 'PAGE SIZE?' and 'INTER-PAGE GAP SIZE?' should total to the number of lines on one page of your forms. If you are outputting to roll paper, or to a CRT terminal such as an H9, then choose two values which provide a pleasing format. If you are using an H8-4 Port, HASL will ask:

BAUD RATE?

HASL asks this question only if it has determined that the specified port is on an H8-4 Interface Card. For other cards, the baud rate is wired onto the board.

When you have completely specified your printer device, HASL asks:

BINARY (Y/N)?

If you do not want a binary (N), the output tape transport is not used and no binary image of the assembled program is placed in memory. Often, no binary is specified until you are sure the program will assemble. If a yes (Y) is given in reply to the question, the assembler then asks

BINARY TAPE (Y/N)?

A no (N) reply to this question directs HASL-8 to place the binary generated from the assembly into memory at the proper location. If NO BINARY TAPE is specified, you should set the HIGH MEMORY limit below the point used by the object program. NOTE: To do this may require reconfiguration of the assembler. See "Appendix A," (Page 4-59).

A yes (Y) reply to this question directs HASL-8 to place the binary generated from the assembly of the source code onto tape at the dump port. This tape is in the memory image format and contains the starting and ending addresses, and the entry point address of the desired program.

Once the assembler determines whether a binary is to be generated or not, and if it is to be placed into memory or dumped onto tape, it then asks

INPUT

The response to this is the character string used to identify the source file when it was created by the Text Editor. Do not include any string delimiters to specify the file name when outputting from Text Editor. For example:

```
NEWOUT "TEST"  
FLUSH  
SURE?
```

dumps a file named TEST using TED-8. It is loaded by the assembler by

```
INPUT?TEST  
FOUND TEST
```

The file name does not have to be complete and can be a null, which allows HASL-8 to load the next file on the tape. (Enter a null by typing a range return.)

Once the file is found, HASL-8 begins the assembly process. The entire file is read for the first pass. Once the first pass is complete, HASL-8 issues the instruction

```
REWIND SOURCE TAPE TYPE CR WHEN DONE.
```

The tape drive is not turned off, so the source tape may be easily returned at its starting point. Once the tape is at its starting point, type a carriage return (CR) and start the tape transport. HASL-8 then issues the instruction

```
FOUND TEST  
POSITION PAPER. TYPE CR:
```

The paper in any printer on the H8 system should be in place and the dump tape transport should be made ready at this time. Position the paper at the top of a form.

Once you type the carriage return, HASL-8 begins the second pass, generating the listing and creating the binary tape. The listing may require reading several records of the input tape and the output binary dump may come in a number of records. Once the listing and the binary dump are complete, HASL-8 terminates its operation by outputting



```
STATEMENTS = xxxxx
FREE BYTES - xxxxx
NO ERRORS DETECTED.      or
```

```
STATEMENTS = xxxxx
FREE BYTES - xxxxx
```

```
ERRORS - xxxxx
```

This first version of this terminating statement indicates that you have successfully completed an assembly of your source program, and if a binary output is specified it is generated. The second version of this terminating statement indicates that you have completed assembly of your source program, but there are errors which the assembler is able to detect. These errors exist in any binary output which may have been specified. Up to three errors per statement line will be shown on the listing output. The errors are shown as single letters in the left hand three columns of the listing. A typical output listing format is shown below.

```
HEATH H8 ASSEMBLER
ISSUE # 4.030.00.
COPYRIGHT HEATH COMPANY, 06/78

.RUBOUT = 00127/21
.
HEATH HASL ISSUE #4.03.00.

LISTING TO PRINTER (Y/N) <N>? N

BINARY (Y/N)? Y
BINARY TAPE (Y/N)? Y
INPUT? USR PROGRAM FOR BASIC 1.0
FOUND USR PROGRAM FOR BASIC 1.0
REWIND SOURCE TAPE. TYPE CR WHEN DONE.
FOUND USR PROGRAM FOR BASIC 1.0
```

HASL #04.03.00

Errors	Addresses	Object Code	Labeler	Opcodes	Operands	PAGE 1 Comments
	117.220			ORG	1200000A-160Q	
	063.207		FPNRM	EQU	063207A	
	117.220	003	START	INX	B	INC UP
	117.221	003		INX	B	TO
	117.222	003		INX	B	EXPONENT
	117.223	012		LDAX	B	(A) = ACCX EXP
	117.224	247		ANA	A	SET CONDX CODE
	117.225	310		RZ		
	117.226	075		DCR	A	/2
	117.227	312 233 077		JZ	USR1	IF UNDER FLOW
	117.232	075		DCR	A	/2 AGAIN (/4)
	117.233	002	USR1	STAX	B	RET TO ACCX
	117.234	315 207 063		CALL	FPNRM	NORMALIZE
	117.237	311		RET		IN CASE 0
	117.240			END	START	

STATEMENTS = 00016
 FREE BYTES - 10331
 NO ERRORS DETECTED.

The previous example illustrated an assembly being listed at the console terminal assigned port address 350Q. The next example outlines the dialog necessary to print a listing on an H14 assigned port address 340Q.

```
LISTING TO PRINTER <NO>? Y
LISTING PORT <340Q>? CR
H14 PRINTER (Y/N) <Y>? CR
PAGE SIZE <60>? CR
```

If you were using an H36 terminal as your printer, the dialog would go as follows:

```
LISTING TO PRINTER <NO>? Y
LISTING PORT <340Q>? 300Q (specify the port for your H36)
H14 PRINTER (Y/N) <N>? CR
PAGE SIZE <60>? CR
INTER-PAGE GAP SIZE <6>? CR
BAUD RATE? 300 (in this case, we are using an H8-4 Port)
```

Errors

All errors detected by the Heath Assembly Language are flagged directly on the listing in the first three columns. One character is flagged for each error detected. If more than one error is detected, the second error character is placed in column 2 and the third error character is placed in column 3.

<u>CHARACTER</u>	<u>ERROR</u>
U	An undefined symbol. The symbol name does not match any symbol in the symbol assignment table. Check for spelling errors or for a completely undefined symbol.
R	Illegal register specified. Two different errors can cause this message. A non-8080 register may have been specified, or the instruction was not meaningful for the register. For example, a register pair instruction which refers to a single register.
D	Label is doubly defined. The symbolic label has been defined twice in the source program.
A	Operand syntax error. The operand expression is improper. For example, it may evaluate to a number >65535, be a divide by zero, or be nonexistant.
V	Value exceeds eight bits. The result of an expression is greater than 255. This error is not flagged if the op-code called for a 16-bit operand such as an LXI instrucion.
F	Format error. A pseudo-op requires a label that is not present in the source code. For example, an EQU pseudo-op requires a label. Too many characters in a label.
O	Unrecognized op-code. The op-code in this statement does not belong to the 8080 instruction set, nor does it belong to the HASL-8 pseudo-op instruction code set. Check for spelling errors or for op-codes used from other microprocessor instruction sets.

CHARACTERERROR

P

Error generated by ERRxx pseudo or reference to a doubly defined label. Note the ERRxx pseudos are generated to flag the user when a test expression does not evaluate satisfactorily.

NOTE: If an assembly generates a great number of errors, it is best to return to the Text Editor, correct as many errors as possible, and reassemble. The reassembly will frequently flag additional errors which are then obvious on the second assembly. If the errors are few, you may load the program and debug it using BUG-8 or PAM-8. However, this **does not** result in a correct listing.

During an Input, one of two error messages may be generated. They are:

SEQ ERR and
CHKSUM ERR.

A sequence error (SEQ ERR) is generated if the file records are not in the proper sequence. For example, if two consecutive label records are read, an error is generated, as a TED-8 Source file consists of a label record followed by text records. The form of the sequence error is

SEQ ERR

Typing a CTRL-C after the SEQ ERR message generates a tape error message

TRY AGAIN?

Reply Y to TRY AGAIN? if you wish to try once more to read the tape. Rewind the tape until you are sure it is before the bad/missing record. HASL-8 will discard all records until the bad/missing one is located. Watch the record numbers on the H8 front panel LED's to make sure you don't miss the record again.

Reply N to TRY AGAIN? if you wish to restart the assembly completely.

A checksum error (CHKSUM ERR) is generated if the actual computed CRC for the record in question does not match the CRC recorded at the start of the record. The form of the checksum error message is

CHKSUM ERR IGNORE?

A Y in response to the question ignore aborts the error message and the next consecutive record is read. NOTE: Do not ignore the checksum error unless there is no other way to recover the data. If a checksum error is flagged, the chances are very good that the data in the designated record is faulty.



Control Characters

CONTROL-C, CTRL-C

CONTROL-C is a general-purpose cancel key. Typing CTRL-C causes HASL-8 to start over at the beginning.

RESTARTS

The H8 front panel keyboard can be used to restart the assembler if control has been returned to PAM-8. HASL-8 can be restarted in two places. They are:

PASS #1 040 100

PASS #2 040 103.

OUTPUT SUSPENSION and RESTORATION, CTRL-S and CTRL-Q

Typing CONTROL-S during an output suspends the output to the terminal and suspends program execution. This command is particularly useful when you use a video terminal, since you can use the CONTROL-S or suspend feature each time a screen is nearly filled and information at the top is about to be lost due to scrolling.

Typing a CONTROL-Q permits HASL-8 to resume execution and outputting information to the terminal. The CONTROL-Q cancels the CONTROL-S function.

The DISCARD FLAG, CTRL-O and CTRL-P

Typing the CONTROL-O toggles the DISCARD FLAG. This stops output on the terminal but does not halt program execution until the program terminates. Typing a CONTROL-P (or retyping CONTROL-O) clears the discard flag. CONTROL-O is often used to discard the remainder of long listings and other similar outputs.

APPENDIX A

Loading Procedures

Loading From the Software Distribution Tape

1. Load the tape in the reader.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. A single beep indicates a successful load.
5. Install any optional patches.
6. Press GO on the H8 front panel.
7. Repeatedly type the space bar on your console until HASL-8 responds:

```
HEATH H8 ASSEMBLER  
ISSUE # 4.03.00.  
COPYRIGHT HEATH COMPANY, 06/78
```

8. Configure HASL-8 as desired, answering the following questions. Prompt each question by typing its first character on the console terminal keyboard.

```
•AUTO NEW-LINE (Y/N)?  
•BKSP=00008/  
•CONSOLE LENGTH=00080/  
•HIGH MEMORY=16383/  
•LOWER CASE (Y/N)  
•PAD=4/  
•RUBOUT=00127/  
•SAVE?  
.
```

9. Before executing SAVE, have the tape transport ready at the DUMP port.
10. To use HASL-8 directly from the distribution tape, type the return key at any time rather than a question prompt key. HASL-8 responds

```
HEATH HASL ISSUE #4.03.00.
```

```
LISTING TO PRINTER (Y/N) <N> Y
```

11. Follow the procedure outlined in "Using the Assembler" on Page 4-51.



Loading From a Configured Tape

1. Load the tape in the tape transport.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. A medium beep indicates a successful load.
5. Press GO key on the H8 front panel.
6. Repeatedly press the space bar on your console.
7. The console terminal responds with:

```
HEATH HASL ISSUE #4.03.00  
LISTING TO PRINTER (Y/N) <N>?
```

HASL-8 is ready to use in the configured form. Proceed to answer the question directing the desired assembly procedure.

INDEX

- Addressing Modes, 4-12
- Arithmetic Instructions, 4-21 ff,
- Assembler Directives, 4-44
- Assembler Operations, 4-51

- Branch Instructions, 4-34 ff,

- Character Set, 4-4
- Character Strings, 4-10
- Comment Field, 4-4, 4-6
- Condition Flags, 4-13
- Conditional Assembly, 4-45
- Control Characters, 4-58

- Data Transfer Instructions, 4-17 ff,
- Define Byte (DB), 4-44
- Define Word (DW), 4-45
- Define Space (DS), 4-44
- Direct, 4-12
- Dollar Sign (\$), 4-4
- Doubly Defined Label, 4-56

- ERRxx, 4-49
- EQU, 4-46
- EJECT, 4-48
- ELSE, 4-45
- END, 4-46
- ENDIF, 4-46
- Errors, 4-56
- Expressions, 4-8

- Format Control, 4-7

- I/O Instructions, 4-38 ff,
- IF, 4-45
- Illegal Register, 4-56
- Immediate, 4-13
- Integers, 4-8

- LOF, 4-49
- LON, 4-49
- Label Field, 4-4 ff,
- Least Significant Bit (LSB), 4-12
- Letters, 4-4
- Listing Control, 4-47

- Logical Instructions, 4-28 ff,

- Machine Control Instructions, 4-38 ff,
- Most Significant Bit (MSB), 4-12

- Numerals, 4-4

- Opcode Field, 4-4 ff,
- OPCODES (8080), 4-11 ff,
 - Arithmetic Group, 4-21 ff,
 - Branch Group, 4-34 ff,
 - Data Transfer Group, 4-17 ff,
 - Logical Group, 4-28 ff,
 - Machine Group, 4-38 ff,
- Operating the Assembler, 4-51
- Operand Field, 4-4, 4-6
- Operator Precedence, 4-8
- Operators, 4-8
- ORG, 4-47
- Origin Symbol (ORG), 4-10, 4-47
- Overflow Error, 4-10

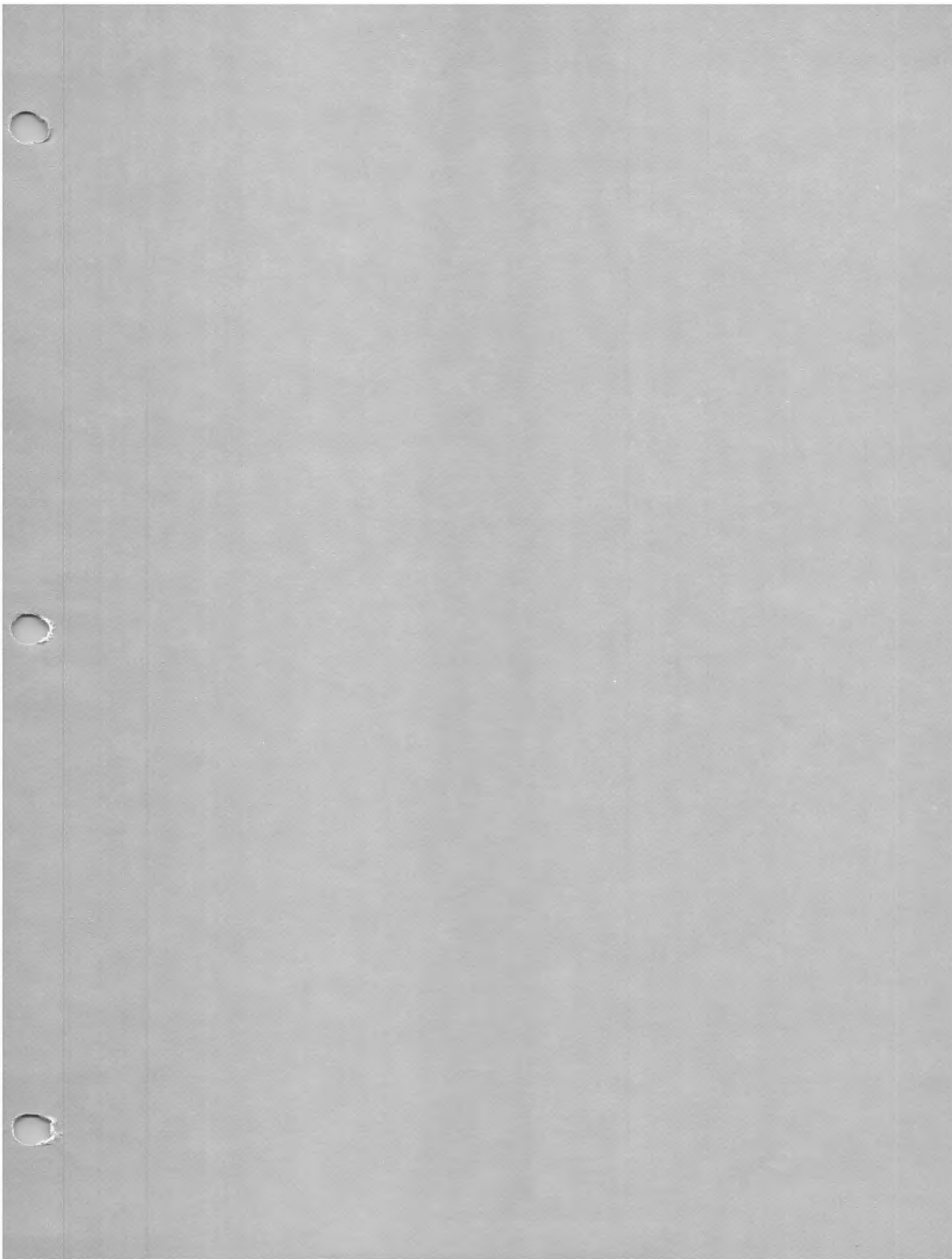
- Period, (.), 4-4
- Pound symbol, (#), 4-9
- Pseudo Opcodes, 4-44

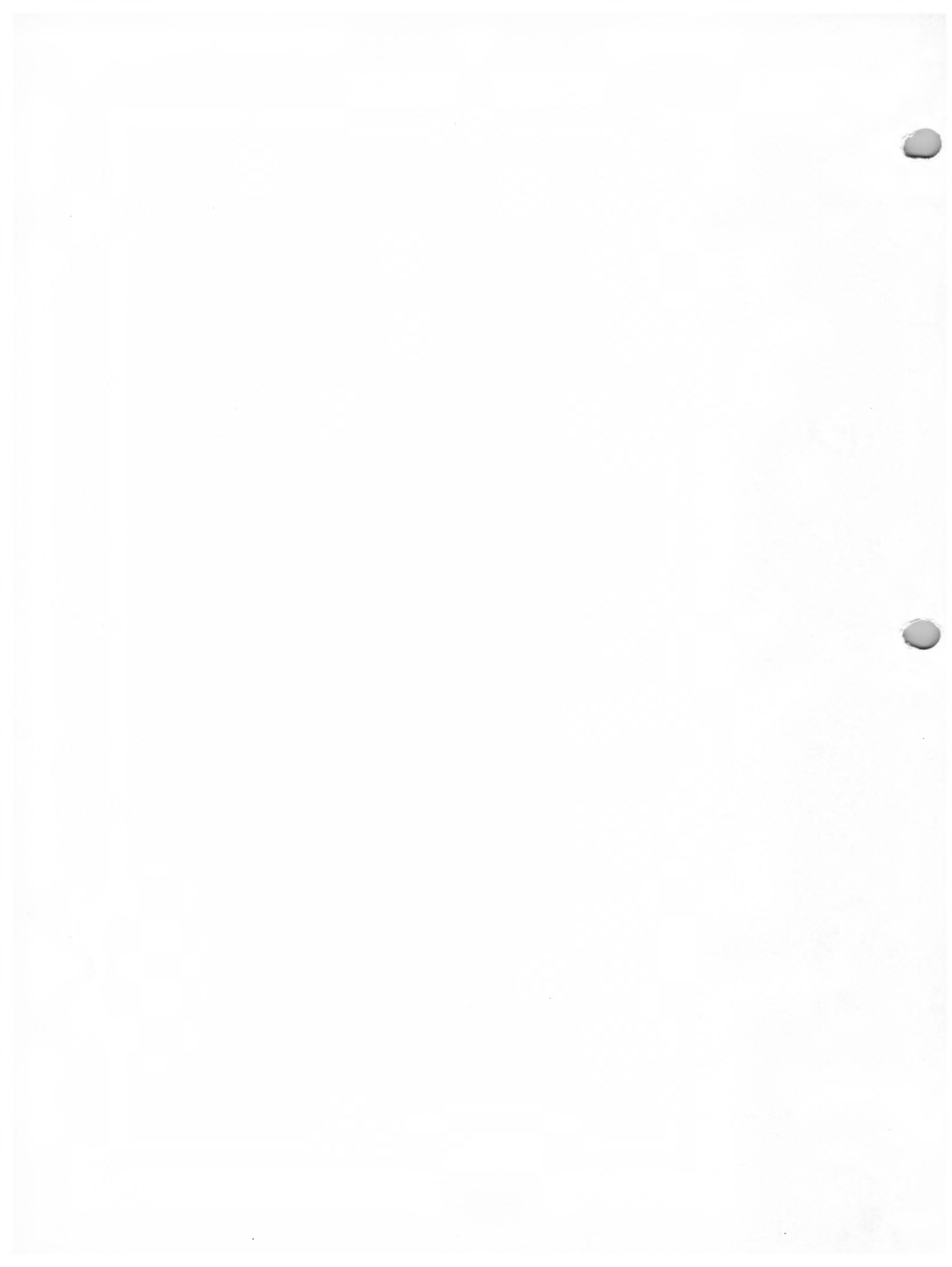
- Register, 4-12
- Register Indirect, 4-12

- Set, 4-47
- Space, 4-48
- Stack Instructions, 4-38 ff,
- Statements, 4-4
- STL, 4-48
- Strings, 4-10
- Symbolic Programs, 4-3
- Symbols, 4-9
- Syntax Error, 4-56

- Text Editor, 4-3
- Title, 4-48
- Tokens, 4-8
- TED-8, 4-3, 4-7

- Undefined Symbol, 4-56
- Unrecognized Op-Code, 4-56





Section 5

BENTON HARBOR BASIC



TABLE OF CONTENTS

INTRODUCTION

Manual Scope	5-5
Hardware Requirements	5-5
Loading and Running BASIC	5-6
Command Completion	5-6

BASIC ARITHMETIC

Data Types	5-7
Variables	5-9
Subscripted Variables	5-10
Expressions	5-12
Arithmetic Operations	5-12
Relational Operators	5-16
Boolean Operators	5-17

THE COMMAND MODE

Using The Command Mode For Statement Execution	5-19
--	------

BASIC STATEMENTS

Line Numbers	5-21
Statement Types	5-21
Command Mode Statements	5-23
Statements Valid In the Command or Program Mode	5-29
Program Mode Statements	5-51

PREDEFINED FUNCTIONS

Introduction	5-55
Arithmetic and Special Feature Functions	5-55

EDITING COMMANDS

Control-C, CTRL-C	5-61
Inputting Control	5-61
Outputting Control	5-62
Command Completion	5-62
Enforced Lexical Rules	5-63
General Text Rules	5-63

ERRORS

Error Messages	5-65
Recovering from Errors	5-65

BASIC ERROR TABLE	5-67
-------------------------	------

APPENDIX A

Loading From the Software Distribution Tape	5-69
Loading From a Configured Tape	5-70
Optional Patches	5-70

APPENDIX B

Numeric Data	5-71
Boolean Data	5-71
Variables	5-71
Subscripted Variables	5-72
Arithmetic Operators	5-72
Relational Operators	5-72
Boolean Operators	5-73
Line Numbers	5-73
The Command Mode	5-73
Multiple Statements on One Line	5-73
Command Mode Statements	5-74
Command and Program Mode Statements	5-75
Program Mode Statements	5-78
Predefined Functions	5-79
Editing Commands	5-81

APPENDIX C

BASIC Utility Routines	5-83
------------------------------	------

APPENDIX D

Entry Points to Utility Routines	5-95
--	------

INDEX	5-97
-------------	------



INTRODUCTION

BENTON HARBOR BASIC is a conversational programming language which is an adaptation of Dartmouth BASIC*. (BASIC is an acronym for Beginners' All Purpose Symbolic Instruction Code.) It uses simple English statements and familiar algebraic equations to perform an operation or a series of operations to solve a problem. BENTON HARBOR BASIC is an interpretive language, compact enough to run in a Heath H8 computer with minimal memory, yet powerful enough to satisfy most problem-solving requirements. The interpretive structure of BASIC affords excellent facilities for the detection and correction of programming errors. It uses advanced techniques to perform intricate manipulations and to express problems more efficiently.

Manual Scope

This Manual is written for the user who is already familiar with the language BASIC. It also describes the implementation of Dartmouth BASIC and, in so doing, provides a brief summary of the language. However, this manual is not intended as an instruction Manual for the language BASIC. If you are not familiar with BASIC, we suggest that you obtain the Heathkit Continuing Education course entitled "Basic Programming," Model EC-1100, before attempting to use this Manual.

Hardware Requirements

Benton Harbor BASIC (B.H. BASIC) requires that your H8 computer system memory contain at least 8 kilobytes of random-access memory. The console driver for B.H. BASIC only supports the H8-5 Serial I/O and Cassette Interface Card. The H8-5 also lets you use a mass storage device, such as a cassette recorder. The device must use the correct port address and baud rate. The "Systems Configuration" section of your manual contains this information. Note that B.H. BASIC does not support an H8-4 I/O Card.

BASIC automatically measures the maximum amount of unbroken memory above the starting point at 8K (40,100 offset octal). It uses all available memory unless the high memory limit is set during the system configuration procedure (see "Product Installation" on Page 0-25 in the Software Reference Manual).

*BASIC is a registered trademark of the Trustees of Dartmouth College.



Loading and Running BASIC

BASIC is distributed in binary format on cassette tape or paper tape. It is loaded in accordance with the software configuration guide, outlined in "Product Installation" on Page 0-23. Once a BASIC system tape is configured, you can load the configured tape, using the internal PAM-8 loader, and start it by pressing the GO key. B.H. BASIC uses the asterisk (*) as a prompt character.

Command Completion

B.H. BASIC employs command completion. This means that BASIC examines each character as you type it on the console keyboard, and when sufficient information is received to uniquely identify one particular command, BASIC finishes typing the command for you. For example, once the letters PR have been typed, the command PRINT is uniquely defined. Therefore, BASIC supplies letters INT and the required blank following the T.

BASIC also watches your spelling. As commands are being typed, letter combinations leading to nonexistent commands are not accepted and the console terminal bell is rung.

BASIC ARITHMETIC

Data Types

BASIC supports two different data types:

1. Numeric data.
2. Boolean data.

NUMERIC DATA

BASIC accepts real and integer numbers. A real number contains a decimal point. BASIC assumes a decimal point **after** integer data. Any number can be used in a mathematical expression without regard to its type. Real numbers must be in the approximate range of 10^{-38} to 10^{+37} . Integer numbers must lie in the range of 0 to 65535. All numbers used in BASIC are internally represented in floating point, which allows approximately 6.9 digits of accuracy. Numbers may be either negative or positive.

In addition to integer and real numbers, BASIC recognizes a third number format. This format, called exponential notation, expresses a value as a decimal number raised to a power of 10. The exponential form is:

$$XXE(\pm)NN$$

where E represents the algebraic statement "times ten to the power of," XX represents up to a six digit integer or real number, and NN represents an integer from 0 to 38. Thus, the number is read as "XX times 10 to the \pm power of NN."

Numeric data in all three forms may be used in the immediate mode, program mode in data statements, or in response to READ and INPUT statements.

Unless otherwise specified, all the numbers including exponents are presumed to be positive.



The results of BASIC computations are printed as decimal numbers if they lie in the range of 0.1 to 999999. If the results do not fall in this range, the exponential format is used. BASIC automatically suppresses all leading and trailing zeros in real and integer numbers. When the output is in exponential format, it is in the form

(±) X . XXXXXE (±) NN

The following are examples of typical inputs and the corresponding output. Note the dropping of leading and trailing zeros, truncation to six places of accuracy, conversion to exponential notation when necessary, and conversion to decimal notation where permitted.

<u>INPUT NUMBER</u>	<u>OUTPUT NUMBER</u>	<u>COMMENTS</u>
0.1	.1	(leading zero dropped)
.0079	7.90000E-03	(<.1 converts to exponential)
0022	22	(leading zeros dropped)
22.0200	22.02	(trailing zeros dropped)
999999	999999	(format maintained)
1000000	1.00000E+06	(converted to exponential)
100000007	1.00000E+08	(truncated to 6 places)
-10.1E+2	-1010	(converted to decimal format)

BOOLEAN VALUES

Boolean values are a subclass of numeric values. Values representing the positive integers from 0-65,535 [$2^{16}-1$] may be used as Boolean data. When using numeric data as Boolean values, the numeric data represents the equivalent 16-bit binary numbers. Fractional parts of numeric data used with Boolean operators are discarded. If the numeric value with the fractional part does not fall into the range of 0-65,535, an illegal number error is generated.

Variables

A BASIC variable is an algebraic symbol representing a number. Variable naming adheres to the Dartmouth specification. That is, variable names consist of one alphabetic character which may be followed by one digit (zero to nine). The following is a list of acceptable and unacceptable variables, and the reason why the variable is unacceptable.

<u>ACCEPTABLE VARIABLES</u>	<u>UNACCEPTABLE VARIABLES</u>	<u>REASON FOR UNACCEPTABILITY</u>
C	2C	A digit cannot begin a variable.
A5	AF	A second character in a variable must be a number (0-9).
D	3	A single number is not an acceptable variable.
L2	\$2	The first character of a variable must be a letter (A-Z).

A value is assigned to a variable when you indicate the value in a LET, READ, or INPUT statement. These operations are discussed in "LET" (Page 5-39), "PRINT" (5-43), and "INPUT" (Page 5-52).

The value assigned to a variable changes each time a statement equates the variable to a new value. The RUN command sets all variables to zero (0). Therefore, it is only necessary to assign an exact value to a variable when an initial value other than zero is required.

Subscripted Variables

In addition to the variables described above, BASIC permits subscripted variables. Subscripted variables are of the form:

$$A_n (N_1, \dots, N_8),$$

where A is the variable letter, n is a number (optional) 0-9, and N_1 thru N_8 are the integer dimensions of the variable. Subscripted variables provide you with the ability to manipulate lists, tables, matrices, or any set of variables. Variables are allowed one to eight subscripts.

The use of subscripts permits you to create multi-dimensional arrays of numeric variables. It is important to note that a dimensioned variable is distinguished from a scalar value of the same name. For example, A and A(N) are distinct variables.

When you are referencing a subscripted variable, each element in the subscript list may consist of an arbitrarily complex expression so long as it evaluates to a numeric value within the allowable range for the indicated dimension. Thus, the subscripted variable A(5,5), would be dimensioned as:

$X = A(2,3)$	is legal
$X = A(2 \uparrow 2, \text{VAL}("4.0"))$	is legal as it is equivalent to A(4,4)

The following are graphic illustrations of simple subscripted variables. In these particular examples, a simple variable (A) is followed by one or two integer expressions in parentheses. For example,

$$A(I)$$

where I may assume the values of 0 to 5, allows reference to each of the six elements $A(0)$, $A(1)$, $A(2)$, $A(3)$, $A(4)$, and $A(5)$. A graphic representation of this 6-element, single-dimension array is shown below. Each box represents a memory location reserved for the value of the variable of the indicated name. Often, the entire array is referred to as A .

$A(0)$
$A(1)$
$A(2)$
$A(3)$
$A(4)$
$A(5)$

NOTE: Subscripted variables begin at zero. Therefore, the previous example 0 to 5 defines six elements.

A two dimensional array $B(I, J)$ allows you to refer to each of the $I \times J$ elements (i.e. $I=4$ and $J=5$) $B(0,0)$, $B(0,1)$, $B(0,2)$, \dots , $B(0,J)$, \dots , $B(I,J)$.

This is graphically illustrated as follows, for $B(3,4)$.

J					
I {	B(0,0)	B(0,1)	B(0,2)	B(0,3)	B(0,4)
	B(1,0)	B(1,1)	B(1,2)	B(1,3)	B(1,4)
	B(2,0)	B(2,1)	B(2,2)	B(2,3)	B(2,4)
	B(3,0)	B(3,1)	B(3,2)	B(3,3)	B(3,4)



BASIC does not presume any dimension. Therefore, the DIMension (DIM) statement must be used to define the maximum number of elements in any array. It is described in “DIM (DIMENSION)” on page 5-30.

Expressions

An expression is a group of symbols to be evaluated by BASIC. Expressions are composed of numeric data, Boolean data, variables, or functions. In an expression, these variables are alone or combined by arithmetic, relational, or Boolean operators.

The following examples show some expressions BASIC recognizes.

ARITHMETIC EXPRESSIONS	BOOLEAN EXPRESSIONS	DESCRIPTION
1.02	255	Data
1.02 + 16	255 OR 003	Combined
A < B		Relational

A major feature of B.H. BASIC is its use of expressions in situations where many other BASICs only permit variables or numbers. This feature permits you to perform sophisticated operations within a particular command or function. It is important to note that not all expressions are valid in a statement. The explanation describing the individual statement lists any limitations.

Arithmetic Operators

BASIC performs multiplication, division, addition, and subtraction. BASIC also supports two unary operators (– and NOT). The asterisk (*) is used to signify multiplication and the slash (/) is used to indicate division.

THE PRIORITY OF ARITHMETIC OPERATIONS

When multiple operations are to be performed in a single expression, an order of priority is observed. The following list shows the arithmetic operators in order of descending precedence. Operators appearing on the same line are of equal precedence.

–(Unary)	(negation)
* /	(multiplication division)
+ –	(addition subtraction)

Parentheses are used to change the precedence of any arithmetic operations, as they are in common algebra. Parentheses receive top priority. Any expression within parentheses is evaluated before an expression without parentheses. The innermost leftmost parenthetical expression has the greatest priority.

UNARY OPERATORS

BASIC supports two unary operators: $-$ and NOT. These operators are referred to as unary because they require only one operand. For example:

```
A = -2
C = NOT D
```

The unary operator $(-)$ performs arithmetic negation. The NOT operator performs Boolean negation. See Page 5-17.

MULTIPLICATION AND DIVISION

BASIC uses the asterisk $(*)$ and the slash $(/)$ as symbols to perform the algebraic operations of multiplication and division respectively. Both multiplication and division require numeric data as operands.

The following examples use the multiplication and division operators:

```
*PRINT 2*6
12

*PRINT 6/3
2

*PRINT 6/3*2
4

*
```

NOTE: This last expression evaluates to 4, not 1; as $*$ and $/$ have equal precedence and therefore the leftmost operator is evaluated first.



ADDITION AND SUBTRACTION

The plus sign (+) and the minus sign (−) perform arithmetic addition and subtraction. The following examples use the plus and minus operators:

```
*PRINT 3
3
*PRINT 3+5
8

*PRINT 10-3
7
```

SUMMARY

In any given expression, BASIC performs arithmetic operations in the following order:

1. Parentheses have top priority. Any expression in parentheses is evaluated prior to a nonparenthetical expression.
2. Without parentheses, the order of priority is:
 - a. Unary minus and NOT (equal priority).
 - b. Multiplication and division (equal priority, proceeds from left to right).
 - c. Addition and subtraction (equal priority, proceeds from left to right).
3. If the rules in either 1 or 2 do not clearly designate the order of priority, the evaluation of expression proceeds from left to right.

The following examples illustrate these principles.

In the first example, the expression $12/6*4$ is evaluated from left to right since multiplication and division are of equal priority:

1. $12/6*4 = 8$
2. $2*4 - 3 = 5$
3. $2*4 - 12/6 = 6$
4. $36 + 6 = 42$

Parentheses may be nested, (enclosed by additional **sets** of parentheses). The expression in the innermost set of parentheses is evaluated first. The next innermost is second, and so on, until all parenthetical expressions are evaluated. For example:

$$6*((8 + 4)/3)$$

Evaluates as:

1. $8+4 = 12$ (addition in parentheses has highest priority).
2. $12/3 = 4$ (next innermost parentheses are evaluated).
3. $6*4 = 24$ (multiplication outside of parentheses is lowest priority).

Parentheses prevent confusion or doubt when you are evaluating the expression. For example, the two expressions

$$D*E+2/4+E/C*A+2$$

$$(D*E)+(2/4)+((E/C)*A))+2$$

are executed identically. However, the second is much easier to understand.

Blanks should be used in a similar manner, as BASIC ignores blanks. The two statements:

```
10 LET B = 3 * 2 + 1
10 LET B=3*2+1
```

are identical. The blanks in the first statement make it easier to read.

Relational Operators

Relational operators compare two variables or expressions. They are generally used with an IF THEN statement. The result of a comparison by the relational operators is either a true or a false. A false is represented by zero, and true is represented by 65535 ($2^{16}-1$). NOTE: These values are chosen so when they are used as Boolean values, false is all zeros and true is all ones.

The following table lists relational operators as used in BASIC.

<u>ALGEBRATIC SYMBOL</u>	<u>BASIC SYMBOL</u>	<u>EXAMPLE</u>	<u>MEANING</u>
=	=	A=B	A is equal to B.
<	<	A<B	A is less than B.
≤	<=	A<=B	A is less than or equal to B.
>	>	A>B	A is greater than B.
>	>=	A>=B	A is greater than or equal to B.
≠	<>	A<>B	A is not equal to B.

The symbols =<, =>, >< are not accepted and BASIC generates a syntax error if they are used.

The following examples show the results of using relational operators.

```
*PRINT 3<4      (true)
65535
```

```
*PRINT 4<3      (false)
0
```

B.H. BASIC differs from most other versions in the use of the relational operator. When you are using BASIC, you may use the relational operators in any expression. When the expression is evaluated, the appropriate numeric answer (0 or 65535) will be used as the answer to that expression.

Boolean Operators

OR

The operator OR performs a Boolean OR on the two integer operands. The integer operands (which must lie in the range of 0 to 65535) are converted to 16-bit binary numbers. The Boolean (logical) 16-bit OR is applied and the result is returned to the equivalent integer representation. NOTE: As the Boolean value chosen to represent true (65535) and false (0), the OR operator implements a standard truth table OR function. For example:

```
*PRINT 132 OR 255      00000000 10000100   132
255                    00000000 11111111   255
                        00000000 11111111   255
```

and

```
*PRINT (3>2) OR (4>9)
65535
```

AND

The AND operator performs a Boolean (logical) AND on the two integer operands. These integer operands must lie in the range of 0 to 65535. The integer operands are converted into 16-bit binary numbers and the logical AND is performed. The result is returned to the equivalent integer representation. NOTE: The AND operator implements a standard AND truth table on the values true (65535) AND false (0). For example:

```
*PRINT 132 AND 255      00000000 10000100   132
132                    00000000 11111111   255
*                        00000000 10000100   132
```

and

```
*PRINT (3>2) AND (9>7)
65535
```

NOT

The NOT operator Boolean negation. That is, the numeric value of the variable is converted into a 16-bit Boolean data value; each **bit** is inverted, and the 16-bit binary number is restored to numeric data. For example:

```
*PRINT NOT 15    0 = 00000000 00001111   and
65520            65520 = 11111111 11110000
*
```



THE COMMAND MODE

Using The Command Mode For Statement Execution

You may solve a problem in BASIC by using a complete program or by use of the **command** mode. **Command** mode makes BASIC an extremely powerful calculator.

Lines of program material entered for later execution are identified by line numbers. BASIC identifies those lines entered for immediate execution by the absence of the line number. That is to say, statements that begin with line numbers are stored, and statements without line numbers are executed immediately when a carriage return is received. For example:

```
10 PRINT "THIS IS AN H8 COMPUTER"
```

is not executed when it is entered at the console terminal. However, the statement:

```
*PRINT "THIS IS THE HEATH H8 COMPUTER"
```

when the RETURN key is typed, immediately writes on the terminal:

```
THIS IS THE HEATH H8 COMPUTER
```

The **command** mode of operation is useful in program de-bugging and performing simple calculations which do not justify the writing of a complete program.

For example, in order to facilitate program de-bugging, you may place STOP statements liberally throughout a program. Once BASIC encounters a STOP statement, the program halts. You can examine and change data values using the **command** mode. The statement

```
CONTINUE
```

is used to continue execution of the program. You can also use the GOSUB and IF commands. Values assigned to variables remain intact using this technique. A SCRATCH, CLEAR, or another RUN command resets these values.



The ability to place multiple statements on a single line is an advantage in the **command** mode. For example:

```
*B = 2:PRINT B:PRINT B + 1
2
3
*
```

Program loops are allowed in the **command** mode. For example, a table of squares can be produced as follows:

```
*FOR A = 1 TO 10:PRINT A,A * A:NEXT A
1          1
2          4
3          9
4         16
5         25
6         36
7         49
8         64
9         81
10        100
*
```

Some statements cannot be used in the **command** mode. The INPUT statement, for example, is not available in the **command** mode, and results in the USE error message. There are certain functions in the **command** mode which make no sense when used in the **command** mode. Statements available in the **command** mode are covered in "Command Mode Statements" on Page 5-23 and "Statements Valid in the Command or Program Mode" on Page 5-29.

BASIC STATEMENTS

A program is composed of one or more lines or “statements” instructing BASIC to solve a problem. Each program line begins with a line number identifying the line and its statement. The line number indicates the desired order of statement execution. Each statement starts with an English word specifying the operation to be performed. Single statements are terminated with the return key. Multiple statements are separated by a colon (:) with the last statement terminated by a return (a non-printing character). A DATA statement cannot share a line with other statements. (See Page 5-47.)

Line Numbers

An integer number begins each line in a BASIC program. BASIC executes the program statements in numerical sequence, regardless of the input order. Statement numbers must lie in the range of 1 to 65,535. It is good programming practice to number lines in increments of 5 or 10 to allow insertion of additional statements when de-bugging the program.

The length of a BASIC statement must not exceed one line. There is no method to continue a statement to a following line. However, multiple statements may be written on a single line. In this situation each statement is separated by a colon. For example:

```
10 PRINT "VALUES",A,A+1 is a single line print statement, whereas
10 LET A=12: PRINT A,A+1,A+2 is a line containing two statements, LET and PRINT.
```

Virtually all statements can be used anywhere in a multiple statement line. There are, however, a few exceptions. They are noted in the discussion of each statement. NOTE: Only the first statement on a line can have a line number. Program control cannot be transferred to a statement **within** a line, but only to the beginning of a line.

Statement Types

BENTON HARBOR BASIC supports three different types of statements. First, there are statements valid only in the command mode. These statements are used for loading programs, erasing memory, and other such functions directing BASIC's activities. Second, there are statements valid as both commands or within a program. Third, there are statements valid only within a program. These statements may not be used in the command mode. Most statements fall into the second category. This means they can appear within a program or be typed directly in the command mode and immediately executed.



As noted earlier, some statements valid in both modes may not be meaningful in both modes.

BASIC is designed to allow maximum versatility in its structure. Thus, almost everywhere that BASIC requires a number, you can use a variable. For example, you can construct a computed GOTO by simply computing a value for a variable, X. The statement

```
GOTO X
```

then redirects the program to the computed line number.

The following three sections are organized as command mode statements, command and program mode statements, and program mode statements. They can be found, respectively in: "Command Mode Statements" Page 5-23, "Statements Valid in the Command or Program Mode" (Page 5-29), and "Program Mode Statements" (Page 5-51).

To simplify some practical descriptions in these sections and those following, the notation below is used to describe allowed expressions:

1. "iexp" indicates an integer expression, an expression lying in the range of 0 to 65535. The fractional part of any integer expression is discarded when the integer is formed.
2. "nexp" indicates a numeric expression. This may be an integer, decimal, or exponential expression with up to 6 decimal places.
3. "sep" indicates a separator. Separators such as the comma and the semi-colon are used to delineate certain portions of BASIC statements.
4. "[]" brackets indicate optional portions of a statement, depending on the exact function desired.
5. "var" indicates a variable.
6. "name" indicates a string used to identify a date, a program, or a language record.

Command Mode Statements

The command mode statements cannot be used within a program. For example, the RUN statement cannot be used within a program to make it self-starting. Any attempt to incorporate one of these statements within a program generates a USE error message.

CONTINUE

CONTINUE begins or resumes the execution of a BASIC program. CONTINUE has the unique feature of not affecting any existing variable values, nor does it affect the GOSUB or FOR stack. CONTINUE is normally used to resume execution after an error in the program or after a CONTROL-C stops the program. CONTINUE may be used to enter a program or a specific line (in conjunction with a GOTO). CONTINUE is unlike RUN, which resets all variables, stacks, etc.. The form of the CONTINUE statement is:

```
CONTINUE
```



In the following example, CONTINUE starts the program at a specific line number.

```
*GOTO 100      (start execution at line 100)
*CONTINUE
```

CONTINUE is also useful for entering a program with a variable or variables set at particular values. For example:

```
*A = 23.5      (Variable A set to 23.5)
*GOTO 230      (start execution at line 230)
*CONTINUE
```

DUMP

The DUMP statement saves the current program text on the mass storage media such as, a cassette tape, connected to the load/dump port. The current program is saved; however, no variables are saved. The specific program name is written with the data so the user may reload the program by the specified name. The form of the DUMP statement is:

```
*DUMP "name"
```

Make the tape drive ready before entering the DUMP statement. BASIC starts the drive, writes the data, and stops the drive. The CONTROL-C can be used to abort the DUMP while in progress. However, if a DUMP is aborted, an incomplete file exists on the tape.

The string "name" may be up to 80 ASCII characters. The normal string ASCII characters are permitted. An example of a DUMP is:

```
*DUMP "STARTREK VER 1.0 03/11/77"
```

This statement dumps the program Startrek version 1.0 dated the 11th of March 1977.

LOAD

The LOAD statement discards the current program in memory. A specified program is loaded from the cassette connected to the load/dump port. The form of the LOAD statement is:

```
LOAD "name"
```

The string "name" may consist of up to 80 ASCII characters. The normal string ASCII characters are permitted. BASIC scans the cassette tape until it finds a program whose name matches the specified string. Before destroying the stored information, the user is asked "SURE?." A "Y" reply causes LOAD to proceed. Any other response cancels LOAD.

NOTE: If the name on the cassette tape is longer than the specified name, a match on the supplied characters in the string "name" is valid. Thus, a program may be dumped with extra information entered in the name such as program version number and data. The program can then be loaded without it. For example:

```
*DUMP "STARTREK VER 1.0 03/11/77"
```

This program, Startrek version 1.0 dated 11 March 1977, may be loaded by

```
*LOAD "STARTREK"  
SURE? Y
```

A match is found between the first eight characters of the DUMP string "STARTREK" and the eight characters of the load command "STARTREK." If a null is used as the load string, the next program on the tape is loaded. Therefore, the statement

```
*LOAD ""
```

loads the next BASIC program appearing on the cassette tape.



The cassette should be made ready before LOAD is executed. BASIC starts and stops the mass storage device. CONTROL-C may be used to abort the load part way through. Use a SCRATCH command to clear the results of an aborted load.

During a load, either one of the following two error messages may be generated:

SEQ ERR and
CHKSUM ERR.

A sequence error (SEQ ERR) is generated if the file records are not in sequence. For example, if two consecutive label records are read an error is generated, as a BASIC file consists of a label file followed by a data file. The form of the sequence error is

SEQ ERR

Type a blank after the SEQ ERR message. This will clear the error. The entire file must be reread.

A checksum error (CHKSUM ERR) is generated if the computed CRC for the record in question does not match the CRC included in the record. The form of the checksum error message is

CHKSUM ERR IGNORE?

A Y in response to the question "ignore" aborts the error message and the next consecutive record is read. Do not ignore the checksum error unless there is no other way to recover the data. If a checksum error is flagged, the chances are very good that the data in the designated record is faulty. If you attempt to use such bad data, it may cause BASIC to crash.

RUN

A prepared program may be executed using the RUN statement. The program is executed starting at the lowest numbered statement. All variables and stacks are cleared (set to zero) before program execution starts.

The form of the RUN statement is:

*RUN

After program completion, BASIC prompts the user with an asterisk (*) in the left margin, indicating that it is ready for additional command statements. If, the program should contain errors, an error message is printed that indicates the error and the line number containing the error, and program execution is terminated. Again, a prompt is given. The program must now be edited to correct the error and rerun. This process is continued until the program runs properly without producing any error messages. See "Errors" (Page 5-65) for a discussion of error messages.

Occasionally a program contains an error that causes it to enter an unending loop. In this case, the program never terminates. The user may regain control of the program by typing CONTROL-C (CTRL-C). This aborts the program and returns control to the user. Storage is not altered in this process. CONTINUE resumes program execution. RUN clears the storage and restarts program execution.

SCRATCH

SCRATCH clears all current storage areas used by BASIC. This deletes any commands, programs, data, or symbols currently stored by BASIC.

SCRATCH should be used for entering a new program from the terminal keyboard to ensure that old program lines are not mixed with new program lines. It also assures a clear symbol table. The form of the SCRATCH statement is:

*SCRATCH

Before destroying stored information, the user is asked "SURE?" A "Y" reply causes SCRATCH to proceed. Any other response cancels SCRATCH. For example:

```
*SCRATCH      (Scratch statement entered.)
SURE? Y       (Are you sure, answer Y (YES,))
*             (BASIC is ready for a new entry.)
```

VERIFY

The VERIFY statement permits you to check a file placed on mass storage without affecting the current program. The VERIFY command responds by indicating the name of the file found, and if the file is correct. A form of the VERIFY command is:

*VERIFY "name"



The string "name" can be the name of the record the user desires to verify or it may be a null (""); in which case, BASIC verifies the first record encountered. For example,

```
*VERIFY "STARTREK"  
FOUND STARTREK VER 1.0 03/11/77  
FILE OK  
*
```

In the above example, the file containing the Startrek dump is verified. Note, that the name of the file is printed immediately as soon as the label record is encountered. The FILE OK message is printed after the data record is read and verified. VERIFY performs a checksum on the contents of all data in the file. Using the VERIFY command does not destroy any program data in memory.

During a VERIFY, one of two error messages may be generated. They are:

```
SEQ ERR and  
CHKSUM ERR.
```

A sequence error (SEQ ERR) is generated if the file records are not in sequence. For example, if two consecutive label records are read an error is generated, as a BASIC file consists of a label file followed by a data file. The form of the sequence error is

```
SEQ ERR
```

Type a blank after the SEQ ERR message. This will clear the error. The entire file must be reread.

A checksum error (CHKSUM ERR) is generated if the computed CRC for the record in question does not match the checksum included in the record. The form of the CRC error message is

```
CHECKSUM ERR - IGNORE?
```


A Y in response to the question (IGNORE?) aborts the error message and the record is considered valid. Do not ignore the checksum error unless there is no other way to recover the data. If a checksum error is flagged, the chances are very good that the data in the designated record is faulty.

NOTE: The command VERIFY is not available if BASIC is patched to use an ASR console terminal as the load dump device.

Statements Valid in the Command or Program Mode

The statements in this section may be used in either the command or the program mode. A few of them have only subtle uses in one mode or the other. Because they may be used in both modes, they are listed in this section.

CLEAR

CLEAR sets the contents of all variables, arrays, buffers, and stacks to zero. The program itself is not affected. The command is generally used before a program is rerun to insure a fresh start if the program is started with a command other than RUN. The form of the CLEAR statement is:

```
*10 CLEAR
```

All variables, arrays, buffers, etc., are cleared before program is executed by RUN. Therefore, a clear statement is not required. However, a program terminated prior to execution (by a STOP command or an error) does not set these variables, etc., to zero. They are left with the last value assigned.

If a section of the program is to be rerun after appropriate editing, the variables, arrays, dimensions, etc., should be reinitialized. You can accomplish this by using the CLEAR statement in the command mode.



DIM (DIMENSION)

The DIMENSION statement explicitly defines the maximum dimensions of array variables. A single dimension array is often called a vector. The form of the DIMENSION statement is:

```
*DIM varname (iexp1[. . . . .,iexpn]) [,varname2 (. . . . .)]
```

The expressions “iexp1” through “iexpn” are integer expressions specifying the bounds of each dimension. Dimensions are 0 to “expn.” So, for example, the statement:

```
DIM A(5,5)
```

reserves an array 6×6 or 36 values. The dimensioned variable presets the values to zero.

You may declare several variables in one DIMENSION statement by separating them with commas. For example:

```
*DIM A6(3,2), B(5,5), C3(10,10)
```

dimensions the following arrays

<u>VARIABLE</u>	<u>SIZE</u>	
A6	4 by 3	12 elements
B	6 by 6	36 elements
C3	11 by 11	121 elements

You can place a DIMENSION statement anywhere in a multiple statement line and it can appear anywhere in the program. However, an array can only be dimensioned once in a program unless it is cleared. DIMENSION statements must be executed before the first reference to the array, although good programming practices place all DIMENSION statements in a group among the first

statements of a program. This allows them to be easily identified and changed if alterations are required. The following example demonstrates the use of the DIMENSION statement with subscripted variables and a two-level FOR statement.

```
*LIST
10 REM DIMENSION DEMO PROGRAM
20 DIM A(5,10)
30 FOR B=0 TO 5
40 LET A(B,0)=B
50 FOR C=0 TO 10
60 LET A(0,C)=C
70 PRINT A(B,C);
80 NEXT C:PRINT :NEXT B
90 END

*RUN
0  1  2  3  4  5  6  7  8  9 10
1  0  0  0  0  0  0  0  0  0  0
2  0  0  0  0  0  0  0  0  0  0
3  0  0  0  0  0  0  0  0  0  0
4  0  0  0  0  0  0  0  0  0  0
5  0  0  0  0  0  0  0  0  0  0

END AT LINE 90
*
```

FOR AND NEXT

FOR and NEXT statements define the beginning and end of a program loop. A program loop is a set of repeated instructions. Each time they are repeated they modify a variable in some way until a predetermined condition is reached, causing the program to exit from the loop. The FOR NEXT statement is of the form:

```
FOR var = nexp1 to nexp2 [STEP nexp3]
NEXT VAR
```



When BASIC encounters the FOR statement, the expressions nex1, nex2 and nex3 (if present) are evaluated. The variable "var" may be a scalar numeric variable, or it may be an element of a numeric array. It is assigned a value of "nex1." For example:

```
*FOR A=2 TO 20 STEP 2:PRINT A;:NEXT A
  2  4  6  8 10 12 14 16 18 20
```

causes the program to execute as long as A is less than or equal to 20. Each time the program passes through the loop, the variable A is incremented by 2 (the STEP number). Therefore, this loop is executed a total of 10 times. When incremented to 22, program control passes to the line following the associated NEXT statement. It is important to note that the initial value used for the variable is the value assigned to the variable expression when it entered the FOR-NEXT loop. For example:

```
*A=10:FOR A=2 TO 20 STEP 2:PRINT A;:NEXT A
  2  4  6  8 10 12 14 16 18 20
*
```

Prior to execution, the variable A is assigned the value 10. The program passes through the loop 10 times. A is assigned the value 20 before exiting the loop.

If "nex2" \geq 0, and the initial value of var \geq "nex2," the loop terminates. For example, the program:

```
*LIST
10 FOR J=2 TO 18 STEP 4
20 J=18
30 PRINT J;:NEXT J
40 END

*RUN
18
END AT LINE 40
*
```

is only executed once, since the value of J = 18 is reached on the first pass, satisfying the termination condition.

A loop created by the statement:

```
*FOR A=20 TO 2 STEP 2:PRINT A::NEXT A
20
*
```

is executed only once, as the initial value exceeds the terminal value. However, if this example is modified to read:

```
*FOR A=20 TO 2 STEP -2:PRINT A::NEXT A
20 18 16 14 12 10 8 6 4 2
*
```

the negative step allows normal operation.

In summary, for positive STEP values, the loop is executed until the variable (var) is greater than the final assigned value (nexp2). For negative STEP values, the loop is executed until the variable (var) is less than the final assigned value (nexp2).

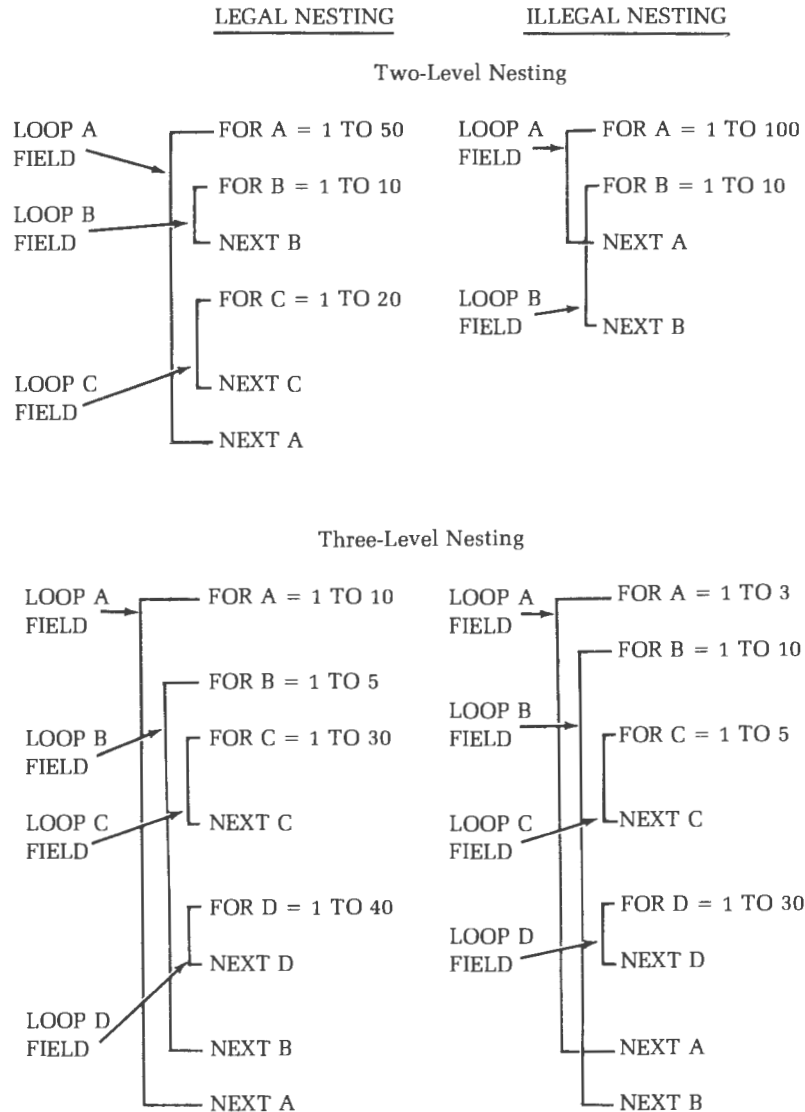
If the loop does not terminate, execution is transferred to the statement following the FOR statement. Therefore, a series of statements may be executed using the incremented value of the variable. If the loop does terminate, execution is transferred to the statement following NEXT.

The expressions in the FOR statement can be any acceptable BASIC numeric expressions.

If the STEP expression and the word STEP are omitted from the FOR statement, a step of +1 is the default value. Since +1 is an extremely common step value, the STEP portion of the statement is frequently omitted. For example:

```
*FOR A=2 TO 10:PRINT A::NEXT A
2 3 4 5 6 7 8 9 10
*
```

Nesting is a technique frequently used in programming. It consists of placing one or more loops completely inside another loop. The field or operating range of the loop (the lines from the FOR statement to the corresponding NEXT statement) must not cross the field of another loop. The following two examples show legal and illegal nesting of FOR NEXT loops.



Note that both columns of nesting illustrations are shown in two-level and three-level forms. However, right-hand columns are not truly nesting but a crossover of FOR and NEXT loops (fields), and therefore are illegal. Also note that each of these examples uses the implied STEP value of 1.

BASIC limits FOR loops to 5 levels. Exceeding 5 levels generates an overflow error.

It is possible to exit from a FOR NEXT loop without reaching the variable termination value. This can be done using a conditional transfer such as an IF statement within the loop. However, control can only be transferred into a loop if the loop is left during prior program execution without being completed. This ensures the assignment of values to the termination and step variables.

Both FOR and NEXT statements can appear anywhere on a multiple statement line.

The NEXT statement does not require the variable. If the variable is not given, BASIC will NEXT the innermost FOR loop.

GOSUB AND RETURN

A subroutine is a section of program performing some operation required one or more times during program execution. Complicated operations on a volume of data, mathematical evaluations too complex for user defined functions, or a previously written routine are all examples of processes best performed by a subroutine.

More than one subroutine is allowed in a single program. Good programming practices dictate that subroutines should be placed one after another at the end of the program in line number sequence. A useful practice is to assign distinctive line number groups to subroutines.

For example, a main program uses line numbers through 300. The 400 block is assigned to subroutine #1 and the 500 block is assigned to subroutine #2. Thus, any errors, program modifications, etc., involving the subroutine are easily identified.



Subroutines are normally placed at the end of a program, but before data statements if there are any.

Program execution begins and continues until a GOSUB statement is encountered. The form of the GOSUB statement is:

```
*GOSUB iexp
```

where *iexp* is the first line in the subroutine. Once GOSUB is executed, program control transfers to the first line of the subroutine and the subroutine is executed. For example:

```
60 GOSUB 500
```

in this example, control (the sequence of program execution) is transferred to line 500 in the program after line 60 is executed. The first line in the subroutine may often be a remark to identify the subroutine, or it may be any executable statement.

Once program control is transferred to a subroutine, program execution continues in the normal line-by-line manner until a RETURN statement is encountered. The RETURN statement is of the form:

```
RETURN
```

RETURN causes program control to return to the statement **following** the original GOSUB statement. A subroutine must always be terminated by RETURN.

Before BASIC transfers control to a subroutine, the next sequential statement to be processed after the GOSUB statement is internally recorded. The RETURN statement draws on this stored information to restart normal program execution. Using this technique, BASIC always knows where to transfer control, no matter how many times subroutines are called.

Subroutines can be nested in the same manner that FOR NEXT statements can be nested. That is, one subroutine can call another subroutine, and if necessary, that subroutine may call a third subroutine, etc. If, during execution of the subroutine a RETURN is encountered, control is returned to the line following the GOSUB calling the subroutine. Therefore, a subroutine can call another subroutine, even

itself. Subroutines can be entered at any point and can have more than one RETURN. Multiple RETURN statements are often necessary when a subroutine contains conditional statements imbedded in it, which cause different subroutine completions dependent on the program data.

It is possible to transfer to the beginning or to any part of the subroutine. Multiple entry points and returns make the GOSUB statement an extremely versatile tool.

Up to 10 levels of GOSUB nesting are permitted in BASIC.

GOTO

The GOTO statement provides unconditional transfer of program execution to another line in the program. The GOTO statement is of the form:

```
*GOTO iexp
```

When this statement is executed, program control transfers to the line number specified by the integer expression "iexp." For example:

```
10 LET A=1
20 GOTO 40
30 LET A=2
40 PRINT A
50 END

*RUN
1

END AT LINE 50
*
```

Program lines in this example are executed in the following order:

10, 20, 40, 50

Line 30 is never executed because the GOTO statement in line 20 unconditionally transfers control to line 40. After the unconditional transfer takes place, normal sequential execution resumes at line 40.

IF THEN (IF GOTO)

The IF THEN (IF GOTO) conditionally transfers program execution from the normal consecutive order of program lines, depending on the results of a relation test. The forms of the IF statement are:

$$\text{IF expression} \left\{ \begin{array}{l} \text{THEN} \\ \text{GOTO} \end{array} \right\} \text{iexp} \quad \text{or}$$

IF expression THEN statement

The expression frequently consists of two variables combined by the relational operators described in "Relational Operators" (Page 5-16). In the first form, if the result of the expression is true, control passes to the specified line number (iexp). In the second form, control passes to the statement following THEN on the remainder of the line. If the result of the expression is false, control passes to the next line. The following examples show use of the IF THEN statement.

```
10 READ A
20 B=10
30 IF A=B THEN 50
40 PRINT "A< >B",A:END
50 PRINT "A=B",A
60 DATA 10,5,20
70 END
```

*RUN

A=B 10

END AT LINE 70

*CONTINUE

A< >B 5

END AT LINE 40

*CONTINUE

A< >B

END AT LINE 40

*

NOTE: The expression can be an arbitrarily complex expression. For example:

```
IF (A<3) AND NOT (B>C) THEN
```

LET

The LET statement assigns a value to a specific variable. The form of the LET statement is:

```
LET var = nexp,
```

However, unlike standard BASIC, multiple assignments are not permitted. For example,

```
LET A=B=3
```

causes A to be set to 65,535 (true) if B is equal to 3, or it causes A to be set to 0 (false) if B is not equal to 3. It does not cause both A and B to be set to 3.

You may omit the key word LET if you prefer. For example, the following two statements produce identical results.

```
10 LET A = 6
```

```
10 A = 6
```

The LET statement is often referred to as an assignment statement. In this context, the meaning of the equal (=) symbol should be understood as it is used in BASIC. In ordinary algebra, the formula $Y = Y + 1$ is meaningless. However, in BASIC the equal sign denotes replacement rather than equality. Thus, the formula $Y = Y + 1$ is translated as add 1 to the current value of Y and store the new result at the location indicated by the variable Y.

Any values previously assigned to Y are combined with 1. An expression such as $D = B + C$ instructs BASIC to add the values assigned to the variables B and C and store the resultant value at the location indicated by the variable D. The variable D is not evaluated in terms of previously assigned values, but only in terms of B and C. Therefore, if previous assignments gave D a different value, the prior value is lost when this statement is executed.

LIST

This command lists the program on the console terminal for reviewing, editing, etc. The form of the list command is:

```
LIST [iexp]
```

Line numbers are indicated by the optional integer expressions. If no line numbers are specified, the entire program is listed. If iexp is specified, BASIC lists the indicated line and the balance of the program lines. You can use a CONTROL-O or CONTROL-C to abort the listing. Refer to "Editing Commands" (Page 5-61) or to "Appendix B" (Page 5-71) for a complete explanation of these functions.

The following are examples of the LIST command.

```
10 LET A=5:LET B=6
20 PRINT A,B,A+B,
30 LET C=A/B
40 PRINT C
50 END
*RUN
5      6      11      .833333

END AT LINE 50
*LIST

10 LET A=5:LET B=6
20 PRINT A,B,A+B,
30 LET C=A/B
40 PRINT C
50 END
*LIST 30

30 LET C=A/B
40 PRINT C
50 END
*
```

ON . . . GOSUB

The ON . . . GOSUB statement allows you to program a computed GOSUB. When you use the ON . . . GOSUB statement, use a RETURN at the end of the subroutine to return program control to the statement **following** the ON . . . GOSUB statement. The form of the ON . . . GOSUB statement is:

```
ON iexp1 GOSUB iexp2, . . . . ., iexpn
```

When it is processing an ON . . . GOSUB statement, BASIC evaluates the expression “iexp1” and uses the result as an index to the list of statement numbers iexp2 thru iexpn. If the expression “iexp1” evaluates to 1, for example, control is passed to the line number given by the expression “iexp2.” If the expression “iexp1” evaluates to 3, for example, control is passed to line number given by the expression “iexp4.” If the expression “iexp1” evaluates to 0, or to an index greater than the number of statement numbers listed, control is passed to the next program statement.

ON . . . GOTO

The ON . . . GOTO statement allows you to perform a computed GOTO. The form of the ON . . . GOTO statement is:

```
ON iexp1 GOTO iexp2, . . . , iexpn
```

When it is processing an ON . . . GOTO statement, BASIC evaluates the expression “iexp1” and uses the result as an index to the list of statement numbers iexp2 thru iexpn. If the expression “iexp1” evaluates to 1, for example, control is passed to the line number given by the expression “iexp2.” If the expression “iexp1” evaluates to 3, for example, control is passed to line number given by the expression “iexp4.” If the expression “iexp1” evaluates to 0, or to an index greater than the number of statement numbers listed, control is passed to the next program statement.

OUT

The OUT statement is used to output binary numbers to an output port. The form of the OUT statement is:

```
OUT iexp1, iexp2
```

The expression “iexp1” is used as the port address, and “iexp2” is the value to be placed at that port. Both iexp1 and iexp2 are decimal numbers. The low-order 8-bits generated by the decimal numbers in iexp1 or iexp2 are used. If you wish to write iexp1 and iexp2 in octal notation for ease in conversion to the actual binary values, write a subroutine or function to perform octal to decimal conversion.

PAUSE

The PAUSE statement causes BASIC to delay before executing the next statement.

```
PAUSE
```

Once the PAUSE statement is executed, no further statements are executed until you type a console terminal character. You can terminate PAUSE by typing any key, and this will not cause the character, to be echoed, but it is good practice to consistently use one character such as space to terminate PAUSE.

WARNING

The POKE function gives an experienced BASIC user direct control of virtually all of the features of the H8 computer. However, subtle misuse of POKE can interfere with the BASIC system and cause it to cease operating correctly. For this reason, HEATH cannot provide consulting support for users who use the POKE function.

POKE

The POKE statement is used to write values into an assigned H8 memory location. The form of the POKE statement is:

```
POKE iexp1, iexp2
```

The low-order 8-bits of iexp2 are inserted into memory location iexp1. NOTE: iexp1 and iexp2 must be given as decimal numbers. If you wish to use octal numbers for ease in referencing to binary notation, you must use a separate octal to decimal subroutine or function to generate these numbers.

PRINT

The PRINT statement is used to output **data** to the console terminal. The form of the PRINT statement is:

```
PRINT [nexp1 sep1 . . . nexpn(sepn)]
```

The expressions and separators contained within the brackets are optional. When used without these optional expressions and separators, the simple PRINT statement outputs a blank line followed by a carriage-return line feed.

Printing Variables

The PRINT statement can be used to evaluate expressions and to simultaneously print their results, or to simply print the results of a previously evaluated expression or evaluations. Any expression contained in the PRINT statement is evaluated before the result is printed. For example:

```
10 A=4:B=6:C=5+A
20 PRINT
30 PRINT A+B+C
40 END
*RUN

19

END AT LINE 40
*
```

All numbers are printed with a preceding and following blank. You can use PRINT statements anywhere in a multiple statement line. NOTE: The terminal performs a carriage-return line feed at the end of each PRINT statement unless you use the separators described in “Use of the , and ;” (Page 5-45). Thus, in the previous example, the first PRINT statement outputs a carriage-return line feed and the second print statement outputs the number 19 followed by a carriage-return line feed.

Printing Strings

The PRINT statement can be used to print a message (a string of characters). The string may be alone or it may be used together with the evaluation and printing of a numeric value. Characters to be printed are designated by enclosing them in quotation marks ("). For example:

```
10 PRINT "THIS IS A HEATH H8"  
*RUN  
THIS IS A HEATH H8  
  
END AT LINE 65535  
*
```

The string contained in a PRINT statement may be used to document the variable being printed. For example:

```
10 LET A=5:LET B=10  
20 PRINT "A + B",A+B  
30 END  
*RUN  
A + B          15  
  
END AT LINE 30  
*
```


When a character string is printed, only the characters between the quotes appear. No leading or trailing blanks are added as they are when a numeric value is printed. Leading and trailing blanks can be added within the quotation marks.

Use of the , and ;

The console terminal is normally initialized with 80 columns divided into five zones. Each zone, therefore, consists of 14 spaces. When an expression in the PRINT statement is followed by a comma (,) the next value to be printed appears in the next available print zone. For example:

```
10 A=5.55555:B=2
20 PRINT A,B,A+B,A*B,A-B,B-A
30 END
*RUN
5.55554          2          7.55554          11.1111          3.55554
-3.55554

END AT LINE 30
*
```

NOTE: The sixth element in the PRINT list is the first entry on a new line, as the five print zones of a 72-character line were used.

Using two commas together in a PRINT statement causes a print zone to be skipped. For example:

```
10 A=5.55555:B=2
20 PRINT @,B,A+B,,A*B,A-B,B-A
30 END
*RUN
5.55554          2          7.55554          11.1111
2.55554          -3.55554

END AT LINE 30
*
```

If the last expression in a PRINT statement is followed by a comma, no carriage-return line feed is given when the last variable is printed. The next value printed (by a later PRINT statement) appears in the next available print zone. For example:

```
10 LET A=1:LET B=2:LET C=3
20 PRINT A,
30 PRINT B
40 PRINT C
50 END
*RUN
1           2
3

END AT LINE 50
*
```

At certain times it is desirable to use more than the designated five print zones. If such tighter packing of the numeric values is desired, a semicolon (;) is inserted in place of the comma. A semicolon does not move the next output to the next PRINT zone, but simply prints the next variable, including its leading and trailing blank. For example:

```
10 LET A=1:LET B=2:LET C=3
20 PRINT A;B;C
30 PRINT A+1;B+1
40 PRINT C+1
50 END
*RUN
1  2  3
2  3
4

END AT LINE 50
*
```

NOTE: If either a comma or a semicolon is the final character in a PRINT statement, no final carriage-return line feed is printed.

READ AND DATA

The READ and DATA statements are used in conjunction with each other to enter data into an executing program. One statement is never used without the other. The form of the statements are:

```
READ var1, . . . , varn
DATA exp1, . . . , expn
```

The READ statement assigns the values listed in the DATA statement to the specified variables var1 through varn. The items in the variable list may be simple variable names or arrays. Each one is separated by a comma. For example:

```
5  DIM A (2,3)
10 READ C, A (1,2)
20 DATA 12, 56
30 PRINT C; A (1,2)
*RUN
12 56

END AT LINE 65535
*
```

Because data must be read before it can be used in the program, READ statements generally occur in the beginning of a program. You may, however, place a READ statement anywhere in a multiple statement line. The type of expression in the DATA statement must match the type of corresponding variable in the READ statement. When the DATA statement is exhausted, BASIC finds the next sequential DATA statement in the program. NOTE: BASIC does not automatically go to the next DATA statement for every READ statement. Therefore, one DATA statement may supply values for several READ statements if DATA statement contains more expressions than the READ statement has variables.

DATA statements may contain arbitrarily complex expressions to represent the data values. Each value expression is separated from other value expressions by a comma. A field in the DATA statement may be left null by means of two adjacent commas. This causes the associated variable to retain its old value. For example:

```
10 A=1:B=1:C=1
20 READ A,B,C
30 PRINT A,B,C
40 DATA 3,,4
50 END
*RUN
3          1          4

END AT LINE 50
*
```



If a DATA statement appears on a line, it must be the only statement on the line. DATA statements may not follow any other statement on the line. Other statements should not follow DATA statements.

DATA statements do not have to be executed to be used. That is, they may be the last statement in a program, and be used by a READ statement executed earlier in the program. However, if DATA statements appear in a program in such a place that they are executed (there are executable statements beyond the DATA statement), the executed DATA statement has no effect. Therefore, location of DATA statements is arbitrary as long as the expressions contained within the DATA statements appear in the correct order. However, good programming practice dictates all DATA statements occur near the end of the program. This makes it easy for the programmer to modify the DATA statements when necessary.

If an expression contained in a DATA statement is bad, the illegal character error message is printed. All subsequent READ statements also cause the message. If there is no data available in the data table for the READ statement to use, the no data error message is printed.

If the number of expressions in the data list exceed those required by the program READ statements, they are ignored, and thus not used.

REM (REMARK)

The REMARK statement lets you insert notes, messages, and other useful information within your program in such a form that it is not executed. The contents of the REMARK statement may give such information as the name and purpose of the program, how the program may be used, how certain portions of the program work, etc.. Although the REMARK statement inserts comments into the program without affecting execution, they do use memory which may be needed in exceptionally long programs.

REMARK statements must be preceded by a line number when used in the program. They may be used anywhere in a multiple statement line. The message itself can contain any printing character on the keyboard and can include blanks. BASIC ignores anything on a line following the letters REM.

RESTORE

The RESTORE statement causes the program to reuse data starting at the first DATA statement. It resets the DATA statement pointer to the beginning of the program. The RESTORE statement is of the form:

```
RESTORE
```

For example:

```
10 READ A,B,C
20 PRINT A,B,C
30 RESTORE
40 READ D,E,F
50 PRINT D,E,F
60 DATA 1,2,3,4,5,6,7,8
70 END
*RUN
1      2      3
1      2      3

END AT LINE 70
*
```

This program does not utilize the last five elements of the DATA statement. The RESTORE command resets the DATA statement pointer and the READ D,E,F, statement uses the first three data elements, as does the initial READ statement.

The CLEAR command includes the RESTORE function.

STEP

The STEP command permits you to step through a program a single line or a few lines at a time. The form of the step command is:

```
STEP iexp
```

where the integer expression iexp indicates the number of lines to be executed before stopping. Execution of the desired lines is indicated by the prompt NXT = nnnn, where nnnn is the next line number to be executed. A STEP 2 is required to execute the first program line. All future single-line executions require a STEP or STEP 1. For example:

```
10 READ A,B,C
20 PRINT A,B,C
30 RESTORE
40 READ D,E,F
50 PRINT D,E,F
60 DATA 1,2,3,4,5,6,7,8
70 END

*CLEAR

*STEP 3
1          2          3
NXT= 30
*STEP
NXT= 40
*STEP
NXT= 50
*STEP
1          2          3
NXT= 60
*STEP 2

END AT LINE 70
*
```

Program Mode Statements

PROGRAM MODE statements are valid only when utilized within a program. If they are entered in the command mode, an illegal use error is flagged.

DATA

The DATA statement discussed in “Read and Data” (Page 5-47) is a program only statement, although it is used in conjunction with a READ statement, which may be used in either the command or program modes.

DEF FN

The DEF FN statement defines single line program functions created by the user. The form of the DEF FN statement is:

```
DEF FN varname (arg1 [,arg2, . . . . ,argn] ) = expr
```

The variable name (varname) must be a legal string or numeric variable name and cannot be previously dimensioned. However, it may be previously defined. The latest definition takes precedence. The argument list “(arg1 [arg2, ,arg3])” must be supplied to indicate a function. NOTE: The arguments are real, not dummy variables, and do change as evaluation proceeds.

```
10 REM DEFINE A SQUARE FUNCTION
20 DEF FN S1(I) = I * I
30 PRINT FN S1(3),I,FN S1(5),I
40 END

*RUN
9          3          25          5

END AT LINE 40
*
```

END

The END statement causes control to return to the command mode. An END statement message is typed, giving the line number of the END statement. END also causes the next statement pointer to be set to the beginning of the program so a CONTINUE resumes execution at the beginning of the program.



An END statement may appear anywhere in the program, as many times as desired. If a program does not contain an END statement, it “runs off the end.” In this case, BASIC generates a pseudo end statement at line 65,535.

INPUT

The INPUT statement is used when data is to be READ from the terminal keyboard or from a mass storage device **working through the console terminal**. The form of the INPUT statement is:

```
INPUT prompt;var1, . . . , varn
```

If the first element in the list following the INPUT statement is a string, INPUT assumes it is a PROMPT and types the string in place of a question mark (?).

Data input from the console terminal has a format identical to the DATA statement.

Expressions may be supplied and null fields cause the variable to retain its previous value. If the user response does not supply sufficient data to complete the INPUT statement, another “?” prompt is issued, requesting more data input at the terminal. **CAUTION:** If you supply too much data, it will be ignored. The next INPUT statement issues a fresh READ to the terminal.

When there are several values to be entered via the input statement, it is helpful to print a message explaining the data needed, using the prompt string. For example:

```
10 INPUT "THE TIME IS";T
20 END
```

When this line of the program is executed, BASIC prints

```
THE TIME IS
```

and then waits for a response.

STOP

The STOP statement causes BASIC to enter the command mode. The message stating the line number of the STOP is printed. The next line pointer is left after the STOP statement, so a CONTINUE statement causes execution to resume on the line immediately after the STOP statement. The STOP statement is of the form:

```
STOP
```

The STOP statement can occur several times throughout a single program with conditional jumps determining the actual end of the program. The following example uses the STOP statement to examine a variable during execution.

```
10 A=1:B=2:C=3
20 PRINT A,B,C
30 END
```

```
*RUN
```

```
1          2          3
```

```
END AT LINE 30
```

```
*15STOP
```

```
*RUN
```

```
STOP AT LINE 15
```

```
*PRINT A
```

```
1
```

```
*15          Stop deleted
```

```
*RUN
```

```
1          2          3
```

```
END AT LINE 30
```

```
*
```



PREDEFINED FUNCTIONS

Introduction

There are 16 predefined functions in B.H. BASIC. These functions perform standard mathematical operations such as square roots, logarithms, and special features. Each function has an abbreviated three- or four-letter name, followed by an argument in parentheses. As these functions are predefined, they may be used throughout a program when required. Predefined functions use numeric expressions (nexp) or integer expressions (iexp). Function key words are automatically followed by an open parenthesis “(”.

The abbreviation (narg) is used to indicate a numeric argument, a decimal number lying in the approximate range of 10^{-38} to 10^{+37} . Certain functions do not permit the argument to assume this wide range, as indicated in the function description.

The predefined functions may be used in either the command or program mode.

Arithmetic and Special Feature Functions

THE ABSOLUTE VALUE FUNCTION, ABS (nexp)

The ABSOLUTE VALUE Function gives the absolute value of the argument. The absolute value is the positive portion of the numeric expression. For example:

```
*PRINT ABS(-5.5)
5.5                or,

*PRINT ABS(SIN(3.5))
.350783
*
```

NOTE: The sine of 3.5 radians is $-.350783$.



THE ARC TANGENT FUNCTION, ATN (nexp)

The ARC TANGENT Function returns the arc tangent of the argument. For example:

```
*PRINT ATN(1/1)*57.296;"DEGREES"
45.0001 DEGREES
*PRINT 4*ATN(1)
3.14159
*                                     NOTE:  $\pi = 3.14159$ 
```

THE COSINE FUNCTION, COS (nexp)

The COSINE function returns the COSINE of the argument (nexp) expressed in radians. For example:

```
*PRINT COS(60/57.296)
.500003
*
```

THE EXPONENTIAL FUNCTION EXP (nexp)

The EXPONENTIAL function returns the value e^{nexp} . If "nexp" exceeds 88, an overflow error is flagged, as the result exceeds 10^{38} . If "nexp" is less than -88, an overflow error occurs. An example of the exponential function is:

```
*PRINT EXP(1),EXP(2),EXP(COS(60/57.296))
2.71828                7.38905                1.64873
*
```

THE INTEGER FUNCTION, INT (narg)

The INTEGER function returns the value of the greatest integer value, not greater than "narg." If the argument is a negative number, the INTEGER function returns the negative number with the same or smaller absolute value. For example:

```
*PRINT INT (38.55)
38

*PRINT INT (-3.3)
-3
```

THE LOGARITHM FUNCTION, LOG (nexp)

The LOGARITHM function returns the natural logarithm (LOG to the base e) of the argument. You can find the Logarithms of a number N in any other base by using the formula:

$$\text{LOG}_a N = \text{LOG}_e N / \text{LOG}_e a$$

where “a” represents the desired base. Most frequently, “a” is 10 when you are converting to common logarithms. For example:

```
*PRINT "A POWER RATIO OF 2 IS";10*(LOG(2)/LOG(10));"DECIBELS"
A POWER RATIO OF 2 IS 3.0103 DECIBELS
*
```

THE PAD FUNCTION, PAD (0)

The PAD function returns the value of the keypad pressed on the H8 front panel. For example:

```
*PRINT PAD(0)
6           The #6 key was pressed.
```

The PAD function uses all the front panel debounce and repeat software contained in PAM-8. (See “The Segment Function,” Page 5-65, for an additional example.)

NOTE: The PAD function must be completely executed before any other function will respond. Therefore, CONTROL-C, etc., will not work until you press an H8 front panel key.

THE PEEK FUNCTION, PEEK (iexp)

The PEEK function returns the numeric value of the byte at memory location iexp.

THE PIN FUNCTION, PIN (iexp)

The PIN function returns the value input from port “iexp” where iexp is a decimal expression ranging from 0-255. For example:

```
*A=PIN(38)
```

Where “A” now contains the data that was at port #38 (46 octal).

THE POSITION FUNCTION, POS (0)

The POSITION function returns the current terminal printhead (cursor) position. Although the numeric argument (0) is ignored, it must be present to complete the function. The value returned is a decimal number indicating the column number of the printhead (cursor) position. For example:

```
*PRINT POS(0), POS(0), POS(0); POS(0); POS(0)
1           15           29   33   37
*
```

THE RANDOM FUNCTION, RND (narg)

The RANDOM number function returns the next element in a pseudo random series. The RANDOM number generator is not truly random, and may be manipulated by controlling the argument. If narg>0, the random number generator

returns the next random number in the series. If $\text{narg} = 0$, the random number generator returns the previously returned random number. If $\text{narg} < 0$, the value "narg" is used as a new seed for a random number, thus starting an entire new series. Using these three inputs to the random number series, the programmer may continuously return the same number while de-bugging the program, determine what the series of numbers will be when the program is run, or start a series of new random numbers each time BASIC is loaded. For example:

```
10 FOR A=0 TO 2
20 PRINT RND(1)
30 NEXT
40 END
```

*RUN

```
.93677
.566681
.53128
```

END AT LINE 40

*RUN

```
.564484
.787262
.332306
```

END AT LINE 40

*20PRINT RND(0)

*RUN

```
.332306
.332306
.332306
```

END AT LINE 40

*20PRINT RND(-1)

*RUN

```
6.25305E-02
6.25305E-02
6.25305E-02
```

END AT LINE 40

*20PRINT RND(-5)

*RUN

```
.460968
.460968
.460968
```

END AT LINE 40

*

THE SEGMENT FUNCTION, SEG (narg)

The SEG function returns a numeric value which is the correct 8-bit binary number to display the digit on the H8 front panel LED's. The argument must be an integer between 0 and 9. The following program demonstrates the use of PAD, POKE, and SEG in B.H. BASIC.

When this program is executed, the H8 front panel LEDs respond to the H8 keypad numeric entries. Therefore, the program uses a POKE command at the first line which stops the display update. The program ends after nine H8 keypad entries.

```
40 POKE 8200,2
50 A=8203
60 FOR I=A TO A+8
70 POKE I,SEG (PAD(0))
80 NEXT I
90 END
```

*RUN

(Press H8 front panel key)

END AT LINE 90

THE SIGN FUNCTION, SGN (narg)

The SGN function returns the value +1 if "narg" is a positive value, 0 if "narg" is 0, and -1 if "narg" is negative. For example:

```
*PRINT SGN(5.6)
1

*PRINT SGN(-500)
-1

*PRINT SGN(12-12)
0

*
```

THE SINE FUNCTION, SIN (nexp)

The SIN function returns the sine of the argument (nexp) expressed in radians. For example:

```
*PRINT SIN(30/57.296)
.499999
*
```



SQUARE ROOT FUNCTION, SQR (narg)

The SQUARE ROOT function returns the square root of "narg." The argument "narg" must be greater than or equal to 0 (for example, positive).

```
*FOR A=0 TO 5:PRINT A,SQR(A),A*A:NEXT
```

0	0	0
1	1	1
2	1.41421	4
3	1.73205	9
4	2	16
5	2.23607	25

*

THE FREE SPACE FUNCTION, FRE (0)

The FREE function returns the number of free bytes of program and variable storage area available. You must supply the dummy argument 0 to complete the function. Most frequently, this function is used in the command mode in conjunction with a PRINT statement. For example:

```
*PRINT FRE (0)
```

```
1224
```

*

THE SPACE FUNCTION, SPC (iexp)

The SPACE function spaces the printhead (cursor) iexp spaces to the right of its present position. For example:

```
*PRINT 12,14,SPC(20);600
```

```
12          14
```

```
600
```

*

THE TAB FUNCTION TAB (iexp)

The TAB function moves the printhead (cursor) to the iexp th column. NOTE: If the printhead is at or past the iexp th column, the function is ignored. For example:

```
*PRINT TAB(20);60,70
```

*

```
60
```

```
70
```


EDITING COMMANDS

BENTON HARBOR BASIC provides several commands used to halt program execution, erase characters, delete lines, add lines, and provide other editing functions. A great number of these editing commands are common to all the Heath H8 Software packages. Their operation is covered in detail in the “Console Driver” section (Page 0-35) of the “Introduction” to this Software Reference Manual.

Control-C, CTRL-C

CONTROL-C is a general-purpose cancel key. It can be used to stop a mass storage input or output operation, stop program execution, stop a listing, and to stop a program during an input statement. Using CONTROL-C results in the CC error message. NOTE: A CONTROL-C causes the program to terminate at the end of a current statement.

Press and hold the CTRL key on your terminal before pressing the C key. This procedure cancels the operation in progress.

Inputting Control

The following control characters take effect when you are inputting information from the terminal.

BACKSPACE, BKSP/CTRL-H

The BACKSPACE key (or a CONTROL-H) causes a one-character backspace. The backspace code is echoed to the terminal so devices with backspace capability physically backspace. Attempting to backspace into column zero is illegal and causes a terminal bell code to be echoed. NOTE: Backspace can be changed at configuration. See “Product Installation” on Page 0-23 of the “Introduction” to this Software Reference Manual.

RUBOUT

The RUBOUT key causes BASIC to discard the current line being inserted. A carriage-return and line feed is sent to the console terminal, and the user may now re-enter the entire line. NOTE: Rubout can be changed at configuration. See “Product Installation” on Page 0-23 of the “Introduction” to this Software Reference Manual.

Outputting Control

The following control characters take effect only when you are printing information to the console terminal. They should not be used when you are inputting information via the console terminal, as they affect characters being echoed to the console.

OUTPUT SUSPENSION AND RESTORATION, CTRL-S AND CTRL-Q

Type CONTROL-S to suspend the output and to suspend program execution. This command is particularly useful when you are using a video terminal; you can use the CONTROL-S (suspend) feature each time a screen is nearly filled and information at the top will be lost due to scrolling.

By typing CONTROL-Q, you permit BASIC to continue execution and outputting information to the terminal. CONTROL-Q cancels the CONTROL-S function.

The DISCARD FLAG, CTRL-O and CTRL-P

Type a CONTROL-O to toggle the DISCARD FLAG. Setting the DISCARD FLAG stops output on the terminal but does not halt program execution until you retype CONTROL-O or until you type a CONTROL-P to clear the DISCARD FLAG. CONTROL-O is often used to discard the remainder of long listings and other similar outputs. BASIC clears the discard flag when it returns to the command mode or when an INPUT statement is executed, so that the prompt will appear.

Command Completion

When you are inputting information from the console terminal in the command mode, or in response to an INPUT command, BASIC checks the incoming characters for the initial characters of a keyword. As soon as enough characters of a keyword are entered to uniquely identify it, and it is distinguished from a variable name, BASIC completes the keyword into the terminal. For example, to enter the command SCRATCH, type SC. Since SC uniquely determines SCRATCH and SC is not a legal variable name, BASIC types the characters RATCH immediately following the SC. Striking the backspace key backspaces over the entire word SCRATCH.

Function keywords are automatically followed by an open parenthesis "(" . Other keywords are immediately followed by a blank.

Enforced Lexical Rules

BASIC enforces two lexical rules during input.

1. Two adjacent alphabetical characters must start a keyword. For example, XX is illegal as no keyword starts with "XX" and "XX" is an illegal variable name. This rule is excepted when following a REMARK statement or when the characters are contained within quotes (indicating a string).
2. A quoted string must be closed; every quote character must have a mate on the same line.

Should a character be typed which is in violation of these rules, a bell code is echoed and the character is ignored.

General Text Rules

BLANKS

BASIC programs are generally "free format." That is, blanks (spaces) may be included freely with the following restrictions.

1. Variable names, keywords, and numeric constants may not contain imbedded blanks.
2. Blanks may not appear before a statement number.

LINE INSERTION

You can insert lines into a BASIC program by simply typing an appropriate line number followed by the desired line of text. This is done in response to the command mode prompt (an asterisk). Except when running a program, BASIC remains in the command mode, showing a single asterisk (*) as a prompt. NOTE: The text should immediately follow the last digit of the line number. Although intervening blanks are allowable, they waste memory. BASIC automatically inserts a blank when listing the text. For example:

```
*100PRINT "HEATH BASIC"  
LIST  
100 PRINT "HEATH BASIC"
```



LINE LENGTH

A line in BENTON HARBOR BASIC is restricted to 80 characters. This restriction on line length is completely independent of the console width, which was established by the software configuration procedure (see Page 0-23 of the "Introduction" to this Software Reference Manual). NOTE: If the console terminal, for example, was set at a width of 34 characters, the console will display three complete lines for one line of BASIC.

LINE REPLACEMENT

Replace existing program lines by simply typing the line number and the new text. This is the same process you use to insert a new line. The old line is completely lost once the new line is entered.

LINE DELETION

Delete lines by typing the line number immediately followed by a carriage-return. You can leave blank lines by typing the single space before typing the carriage-return.

ERRORS

BASIC detects many different error conditions. When an error is detected, a message of the form:

```
! ERROR-(ERROR MESSAGE) [at line NNNNN]
```

is typed. BASIC returns to the command mode (if it is not already in the command mode), ringing the console terminal bell. If BASIC is in the command mode, the “at line NNNN” portion of the error message is omitted. For example:

```
*PRINT 1/0
! ERROR - /0
*10PRINT 1/0
*RUN

! ERROR - /0 AT LINE 10
*
```

NOTE: If a line of BASIC contains an error, you can correct it by retyping the entire line. Once the line number is typed, the contents of the old line are lost. To delete a line, type the line number and follow it with a carriage-return.

Error Messages

The following “Basic Error Table” describes the error messages generated by BENTON HARBOR BASIC. An explanation is given after each error message in the Comments column.

Recovering from Errors

When an error is detected, BASIC enters the command mode with the variables and stacks as they were at the time of the error. Thus, the user can use PRINT and LET statements to examine and alter variable contents. Likewise, a GOTO statement can be used to set the pointer to any desired statement number. Often, a combination of these techniques allows the user to continue a program with the error corrected.

NOTE: If you modify text in a B.H. BASIC program, B.H. BASIC CLEARS all variables and you must restart the entire program.



BASIC ERROR TABLE

BASIC	COMMENTS
AC	Argument count. Incorrect number of arguments supplied to a DEF defined function.
CC	CONTROL-C. Program execution or other operation aborted by a CONTROL-C typed at the console terminal.
DE	Data exhausted. A READ called for more data than is available in the DATA statement.
/0	Divide by zero. An attempt to divide by zero. NOTE: Either dividing by the number zero or dividing by an expression which evaluates to zero generates this error.
IN	Illegal number. The line number referenced by a command or program statement is not used by BASIC.
IU	Illegal usage. A command or statement is used in the improper context.
NX	Next variable missing. No FOR statement matching the accompanying NEXT statement.
OV	Overflow. Memory space is filled by program text.
RE	Return error. A RETURN is encountered without a calling statement.
S#	Statement number. The referenced statement number does not exist in the program.

**BASIC****COMMENTS**

SY	Syntax error. Command, statement, or function uses incorrect separators, functions, etc..
MO	Table overflow. One of the internal tables has grown too large for memory. Check GOSUBs, FOR loops, and DIM.
SR	Subscript range. The subscript size of a dimensioned variable exceeded the size defined by the DIM statement.
SC	Subscript count. The number of subscripts assigned to a variable exceeds the number defined in the DIM statement.
ND	Not dimensioned. The subscripted variable has not been dimensioned in a DIM statement.
IC	Illegal character. An improper character is assigned to a command or function NOTE: In B.H. BASIC, an attempt to assign a string to a numeric variable results in an illegal character message.
FU	Function error. No single line function defined by a DEF statement was found when the FN function was encountered. NOTE: The DEF FN must be executed prior to executing the FN function.
TE	Tape error. An error in handling the mass storage device at the load dump port.

APPENDIX A

Loading Procedures

Loading From the Software Distribution Tape

1. Load the tape in the reader.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. A single beep indicates a successful load.
5. Use PAM-8 to install any optional patches.
6. Press GO on the H8 front panel.
7. The console terminal will respond with:

```
HEATH/WINTEK H8 BASIC
BENTON HARBOR BASIC ISSUE # 05.01.02
COPYRIGHT 01/77 BY WINTEK CORP.
```

8. Configure B.H. BASIC as desired, answering the following questions. Prompt each question by typing its first character on the console terminal keyboard. (See "Product Installation" on Page 0-23 in the Introduction to this software Reference Manual.)

```
•AUTO NEW-LINE (Y/N)?
•BKSP = 00008/
•CONSOLE LENGTH = 00080/
•HIGH MEMORY = 16383/
•LOWER CASE (Y/N)?
•PAD = 4/
•RUBOUT = 00127/
•SAVE?
•
```

9. If you execute SAVE, have the tape transport ready at the DUMP port.
10. To use BASIC directly from the distribution tape, type the return key at any time. The Console Terminal will display:

```
B.H. BASIC # 05.01.02
```

```
*
```

BASIC is ready to use.



Loading From a Configured Tape

1. Load the tape in the tape transport.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. A single beep indicates a successful load.
5. Press GO on the H8 front panel.
6. The console terminal responds with:

```
B.H. BASIC # 05.01.02
```

```
*
```

BASIC is ready to use in the configured form.

Optional Patches

“Appendix A” in the Introduction to this Software Reference Manual (Page 0-33) outlines a procedure to install patches.

```
BENTON HARBOR BASIC
05.01.xx.
```

```
OPTION PATCH #1
```

```
2 STOP BITS
```

USE: This patch is inserted for systems which use a terminal device requiring 2 stop bits. This should not be used for devices which can run with only one stop bit.

NOTES: None.

```
-----
```

```
041010 316
075201 001
```

```
-----
```

APPENDIX B

A Summary of BASIC

For additional details, refer to the page number that is given with each of the following topics.

See Page

Numeric Data

5-7

Numbers may be real or integer with the following characteristics:

Range 10^{-38} to 10^{+37} .
Accuracy 6.9 digits.
Decimal range 0.1 to 999999.
Exponential format $(\pm) X.XXXXXE (\pm) NN$.

Boolean Data

5-8

Integer numbers from 0 to 65535 represent two byte binary data from 00000000 00000000 to 11111111 11111111. Fractional parts of numbers between 0 and 65535 are discarded.

Variables

5-9

Variables are named by a single letter (A through Z), or a single letter followed by a single number (0 through 9). For example: A or A6.



See Page

Subscripted Variables

5-10

Subscripted variables are named like variables, but are followed by dimensions in parentheses. Subscripted variables are of the form:

$A_n(N_1, N_2, \dots, N_x)$ For example: $A(1, 2, 7)$ or $A_6(1, 5)$.

You must use a DIMENSION statement to define the range and number of allowable subscripts for a variable.

Arithmetic Operators

5-12

Listed in order of priority. Operators on the same line have equal precedence. Parenthetical operations are performed first. Precedence is left to right if all other factors are equal.

<u>SYMBOL</u>	<u>EXPLANATION</u>
-	Unary negation logical compliment
* /	Multiplication division
+ -	Addition subtraction

Relational Operators

5-16

<u>SYMBOL</u>	<u>EXPLANATION</u>
=	Equal to
<	Less than
< =	Less than or equal to
>	Greater than
> =	Greater than or equal to
< >	Not equal to

See Page

Boolean Operators

5-17

Boolean operators perform the Boolean (logical) operations on two integer operands. The operands must evaluate to integers in the range of 0 to 65535. The operators are:

NOT	Logical complement, bit by bit
OR	Logical OR, bit by bit
AND	Logical AND, bit by bit

Line Numbers

5-21

When it is used in the program mode, BASIC requires that each line be preceded by an integer line number in the range 1 to 65535.

The Command Mode

5-19

The command mode does not use line numbers. Statements are executed when a carriage-return is typed.

Multiple Statements on One Line

5-21

BASIC permits multiple statements on one line. Each statement is separated from the others by a colon (:). DATA statements may not appear on lines with other statements.

*See "Basic Statements."



Command Mode Statements

<u>COMMAND</u>	<u>FORM</u>	<u>DESCRIPTION</u>	<u>See Page</u>
CONTINUE	CONTINUE	Resumes program execution.	5-23
DUMP	DUMP "name"	Saves current program "name" on mass storage media at load dump port; "name" is up to 80 ASCII characters.	5-24
LOAD	LOAD "name"	Loads program "name" from mass storage media at load dump port; "name" is up to 80 ASCII characters. Current program is destroyed.	5-25
RUN	RUN	Start execution of current program. Preclears all variables, stacks, etc..	5-26
SCRATCH	SCRATCH SURE?Y	Clears all program and data storage area. Any response to SURE but Y cancels SCRATCH.	5-27
VERIFY	VERIFY "name"	Performs a checksum on the mass storage record titled "name." No response if record is bad.	5-27

Command and Program Mode Statements

<u>COMMAND</u>	<u>FORM</u>	<u>DESCRIPTION</u>	<u>See Page</u>
CLEAR	CLEAR [<i>varname</i>]	Clears all variables, arrays, buffers, etc.	5-29
DIMENSION	DIM <i>varname</i> (<i>iexp1</i> [, , <i>iexpn</i>]) [, <i>varname2</i> (. . . .)]	Defines the maximum size of variable arrays.	5-30
FOR/NEXT	FOR <i>var</i> = <i>nexp1</i> TO <i>nexp2</i> [STEP <i>nexp3</i>] NEXT <i>var</i>	Defines a program loop. <i>Var</i> is initially set to <i>nexp1</i> . Loop cycles until NEXT is executed; then <i>var</i> is incremented by <i>nexp3</i> (default is +1). Looping continues until <i>var</i> > <i>nexp2</i> (or less than <i>nexp2</i> if STEP is negative). The statement after NEXT is then executed.	5-31



<u>COMMAND</u>	<u>FORM</u>	<u>DESCRIPTION</u>	<u>See Page</u>
GOSUB/ RETURN	GOSUB iexp RETURN	Transfers execution sequence of program to line iexp (the beginning of a subroutine). RETURN returns execution sequence to the statement following the calling GOSUB.	5-35
GOTO	GOTO iexp	Unconditionally transfers the program execution sequence to the line iexp.	5-37
IF/THEN	IF expression THEN iexp IF expression THEN statement	If the expression is true, control passes to iexp line or to "statement." If the relation is false, control passes to the next independent statement.	5-38
LET	LET var = nexp	Assigns the value nexp to the variable. The LET keyword is optional.	5-39
LIST	LIST[iexp]	Lists the entire program on the console terminal. Lists the line iexp through the end of your program.	5-40
ON/GOSUB	ON iexp1 GOSUB iexp2, . . . ,iexpn.	Permits a computed GOSUB. iexp1 is evaluated and acts as an index to line numbers iexp2 thru iexpn, each pointing to a different subroutine.	5-40

<u>COMMAND</u>	<u>FORM</u>	<u>DESCRIPTION</u>	<u>See Page</u>
ON/GOTO	ON iexp1 GOTO iexp2, . . . ,iexpn	Permits a computed GOTO. Iexp1 is evaluated and acts as an index to line numbers iexp2 thru iexpn.	5-41
OUT	OUT iexp1, iexp2	Outputs a number iexp2 to output port iexp1.	5-41
PAUSE	PAUSE	Ceases program execution until a console terminal key is typed.	5-42
POKE	POKE iexp1, iexp2	Writes a number iexp2 into memory location iexp1.	5-42
PRINT	PRINT(nexp. sep1 . . . nexpn(sepn)	Prints the value of the expressions exp with a leading and trailing space. Expressions may be numeric or string. If the separator is a comma, the next print zone is used. If the separator is a semicolon, no print zones are used. No separator prints each expression value on a new line.	5-43
READ & DATA	READ var1, . . . ,varn DATA exp1 . . . ,expn	The READ statement assigns the values exp1 thru expn in the data to the variables var1 thru varn.	5-47



<u>COMMAND</u>	<u>FORM</u>	<u>DESCRIPTION</u>	<u>See Page</u>
REMARK	REM	Text following the REM is not executed and is used for commentary only.	5-48
RESTORE	RESTORE	Causes the program to reset the DATA pointer, thus reusing data at the first DATA statement.	5-49
STEP	STEP iexp	Executes iexp lines of the program. Then returns BASIC to the command mode.	5-50

Program Mode Statements

DEF	DEF FN varname (arg list) = exp	Defines a single-line program function created by the user.	5-51
END	END	Causes control to return to the command mode.	5-51
INPUT	INPUT prompt; var1, ,varn	Reads data from the console terminal. <i>String data must be enclosed in quotes.</i>	5-52
STOP	STOP	Causes BASIC to enter the command mode when the statement containing STOP is executed.	5-53

Predefined Functions

<u>FUNCTION</u>	<u>DEFINITION</u>	<u>See Page</u>
ABS (nexp)	Returns the absolute value of nexp.	5-55
ATN (nexp)	Returns the arctangent of nexp (radians).	5-56
COS (nexp)	Returns the cosine of nexp (radians).	5-56
EXP (nexp)	Returns e^{nexp} .	5-56
INT (narg)	Returns the integer value of narg.	5-56
LOG (nexp)	Returns the natural logarithm of nexp.	5-56
PAD (0)	Returns the value of the H8 front panel key pressed. Includes key debounce.	5-57
PEEK (iexp)	Returns the numeric value at memory location iexp.	5-57
PIN (iexp)	Returns the data input from port iexp.	5-57
POS (0)	Returns the current terminal printhead (cursor) position (by column number).	5-57
RND (narg)	Returns a random number. If narg > 0, RND is next in the series. If narg = 0, RND is the previous random number. If narg < 0, RND algorithm uses narg as a new seed.	5-57
SEG (narg)	Returns the correct eight-bit number to display narg (0-9) on the H8 LEDs.	5-59
SGN (narg)	Returns +1 if narg is positive. Returns -1 if narg is negative. Returns 0 if narg is zero.	5-59
SIN (nexp)	Returns the sine of nexp (radians).	5-59
SPC (iexp)	Positions printhead (cursor) iexp columns to the right.	5-60
SQR (narg)	Returns the square root of narg.	5-60



<u>FUNCTION</u>	<u>DEFINITION</u>	<u>See Page</u>
TAB (iexp)	Position printhead (cursor) to the iexp th column.	5-60
FRE (0)	Returns the amount of free memory in B.H. BASIC.	5-60

Editing Commands

<u>COMMAND</u>	<u>FUNCTION</u>	<u>See Page</u>
CONTROL-C	General-purpose cancel. Returns BASIC to monitor mode from any operation or program execution.	5-61
CONTROL-S	Suspends the output to the console terminal and suspends program execution.	5-62
CONTROL-Q	Restores the output to the console terminal and restores program execution.	5-62
CONTROL-O	Toggles the output discard flag. Does not stop program execution.	5-62
CONTROL-P	Clears the discard flag set by CONTROL-O.	5-62
CONTROL-H	Causes a one-character backspace.	5-61



APPENDIX C

Basic Utility Routines

The following pages contain a description of several utility routines included in BASIC. They can be used with user-written machine language routines called by the USR function. See “Appendix D” for Utility Routine entry points.

BASIC - WINTEN BASIC INTERPRETER.
 SELECTED SOURCE LISTING.

HEATH X8ASM V1.0 02/18/77
 13:12:38 01-APR-77 PAGE 1

000,000 2 XTEXT MTR

74 *** BASIC - *WINTEN* BASIC INTERPRETER.

75 *

76 * J. G. LETWIN, 09/76, FOR *WINTEN* CORPORATION.

77 *

LAFAYETTE, IN.

79 *** COPYRIGHT 09/1976, *WINTEN* CORPORATION,

80 *

902 N. 9TH ST.

81 *

LAFAYETTE, IN. 47901

83 ** LOW-MEMORY CELLS USED BY BASIC

84

040,064

85

ORG 7*3+,UIVED

86

040,064

87

DS 2

ACCX TYPE

040,066

88 ACCX

DS 4

040,072

89

DS 2

ACCY TYPE

040,074

90 ACCY

DS 4

91

92


```

.....
BASIC - WINTER BASIC INTERPRETER.
.....
USR - PERFORM USER ASSEMBLY LANGUAGE FUNCTION,
.....
HEATH XBASH V1.0 02/18/77
13112140 01-APR-77 PAGE 2
.....

```

```

.....
95 **      USR - CALL USER ASSEMBLY LANGUAGE FUNCTION.
96 *
97 *      THE *USR* FUNCTION IS ACTUALLY A CALL TO A USER-WRITTEN ROUTINE
98 *      WHICH MUST HAVE BEEN PREVIOUSLY LOADED INTO MEMORY BY THE USER.
99 *
100 *     THE ADDRESS OF THE FUNCTION'S ENTRY POINT MUST BE IN *USRFCN*.
101 *
102 *     BASIC MUST HAVE BEEN PREVIOUSLY CONFIGURED SO THAT THE USER
103 *     FUNCTION RESIDES IN MEMORY ABOVE THE STACK POINTER (HIGH MEMORY)
104 *     OR ELSE BASIC WILL OVERLAY THE FUNCTION WITH DATA.
105 *
106 *     THE FUNCTION IS ENTERED WITH A POINTER TO THE SINGLE ARGUMENT (IN
107 *     FLOATING POINT), AND MAY RETURN A FLOATING POINT OR STRING ARGUMENT.
108 *     IF NO RETURN VALUE IS PLACED IN *ACCX*, THEN THE ORIGINAL ARGUMENT
109 *     REMAINS THERE, AND USR( RETURNS ITS ARGUMENT AS ITS VALUE.
110 *
111 *     ENTRY   (BC) = #ACCX
112 *     EXIT    (ACCX) CONTAINS VALUE
113 *     USES    ALL
114
115
116
117 USRFCN  DW      0      ADDRESS OF USER FUNCTION ENTRY POINT
118 *
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
```

BASIC - WINTER BASIC INTERPRETER.
ERROR PROCESSING

HEATH X8ASM V1.0 02/18/77
13:12:41 01-APR-77 PAGE 3

```

121 **      ERROR PROCESSING.
122 *
123 *      THESE ERROR PROCESSORS ARE ENTERED WHEN AN ERROR IS DETECTED.
124 *
125 *      CONTROL PASSES DIRECTLY BACK TO COMMAND MODE.
126
127
128 ERR.CC EQU *      CONTROL-C
129
130 ERR.CB EQU *      CNTRL-B
131
132 ERR.DE EQU *      DATA EXHAUSTED
133
134 ERR.DO EQU *      /O
135
136 ERR.IN EQU *      ILLEGAL NUMBER
137
138 ERR.IU EQU *      ILLEGAL USAGE
139
140 ERR.NV EQU *      NEXT VARIABLE MISSING
141
142 ERR.OV EQU *      OVERFLOW
143
144 ERR.RE EQU *      RETURN ERROR
145
146 ERR.SL EQU *      STRING LENGTH
147
148 ERR.SN EQU *      STATEMENT NUMBER
149
150 ERR.SY EQU *      SYNTAX ERROR
151
152 ERR.TC EQU *      TYPE CONFLICT
153
154 ERR.TO EQU *      TABLE OVERFLOW
155
156 ERR.SR EQU *      SUBSCRIPT RANGE
157
158 ERR.SC EQU *      SUBSCRIPT COUNT
159
160 ERR.ND EQU *      NOT DIMENSIONED
161
162 ERR.IC EQU *      ILLEGAL CHARACTER
163
164 ERR.UD EQU *      UNDEFINED FUNCTION
165
166 ERR.TF EQU *      TAPE ERROR

```

```

169 **      CVX - COPY VALUE INTO 'X' ACCUMULATOR.
170 *
171 *      CVX COPIES A 4 BYTE VALUE INTO THE X ACCUMULATOR.
172 *
173 *      ENTRY  (DE) = ADDRESS OF VALUE
174 *      EXIT   COPIED
175 *      USES   A,F
176
177
178 CVX EQU *
```

```

180 **      CXY - COPY (ACCX) INTO (ACCY)
181 *
182 *      ENTRY  NONE
183 *      EXIT   NONE
184 *      USES   A,F,D,E
185
186
187 CXY EQU *
```

```

189 **      CXV - COPY X TO VALUE.
190 *
191 *      CXV COPIES THE CONTENTS OF THE 'X' ACCUMULATOR INTO A MEMORY
192 *      LOCATION.
193 *
194 *      ENTRY  (DE) = TARGET ADDRESS
195 *      EXIT   COPIED
196 *      USES   A,F
197
198
199 CXV EQU *
```

```

201 **      IFIX - SPLIT NUMBER INTO INTEGER AND FRACTION.
202 *
203 *      IFIX FIXES ((DE)) INTO AN INTEGER.
204 *
205 *      ENTRY  (DE) = ADDRESS OF NUMBER
206 *      EXIT   (DE) = INTEGRAL PART OF 0<N<=65535
207 *      TO ERR. IN OTHERWISE
208
209
210 IFIX EQU *
```

BASIC - WINTER BASIC INTERPRETER.
UTILITY SUBROUTINES

HEATH X8ASM V1.0 02/18/77
13:12:43 01-APR-77 PAGE 5

```

212 **      IFLT - FLOAT NUMBER.
213 *
214 *      ENTRY (DE) = VALUE
215 *      EXIT  (ACCX) = NUMBER VALUE
216 *      (DE) = #ACCX-1
217
218
219 IFLT EQU *
```

```

221 **      TDI - TYPE DECIMAL INTEGER.
222 *
223 *      TDI TYPES AN INTEGER AS A 5 PLACE NUMBER. LEADING ZEROS ARE
224 *      SUPPRESSED.
225 *
226 *      ENTRY (DE) = NUMBER
227 *      EXIT  TYPED
228 *      USES  A,F,D,E
229
230
231 TDI EQU *
```

```

233 **      XCY - EXCHANGE (ACCX) WITH (ACCY)
234 *
235 *      ENTRY NONE
236 *      EXIT  NONE
237 *      USES  A,F
238
239
240 XCY EQU *
```

```

242 **      ZRO - ZERO MEMORY.
243 *
244 *      ZRO ZEROS A FIELD OF MEMORY.
245 *
246 *      ENTRY (HL) = ADDRESS
247 *      (DE) = COUNT
248 *      EXIT  NONE
249 *      USES  A,F,D,E,H,L
250
251
252 ZRO EQU *
```

```

255 **      H8 FLOATING POINT FORMAT:
256 *
257 *      SINGLE-PRECISION FLOATING POINT NUMBERS ARE REPRESENTED
258 *      BY A 4-BYTE VALUE. THE NUMBER CONSISTS OF A 3-BYTE
259 *      TWO'S COMPLEMENT MANTISSA, AND A ONE-BYTE BIASED BINARY
260 *      EXPONENT. THE NUMBER FORMAT IS:
261 *
262 *
263 *      N+0      LEAST SIGNIFICANT MANTISSA BYTE
264 *      N+1      MID      SIGNIFICANT MANTISSA BYTE
265 *      N+2      MOST SIGNIFICANT MANTISSA BYTE
266 *      N+3      BIASED EXPONENT
267 *
268 *
269 *      EXPONENT:
270 *      -----
271 *
272 *      EACH FLOATING POINT NUMBER CONTAINS A BINARY EXPONENT.
273 *      THE EXPONENT CONTAINS A BIAS OF 128 (200 OCTAL).
274 *
275 *      THUS AN EXPONENT OF 0 IS ENTERED AS 200Q, AN EXPONENT OF -10
276 *      IS CODED AS 166Q, ETC. THE NUMBER ZERO IS TREAATED AS A
277 *      SPECIAL CASE: ITS EXPONENT (AND MANTISSA) IS ALWAYS 0.
278 *
279 *
280 *      MANTISSA:
281 *      -----
282 *
283 *      THE MANTISSA OCCUPYS 3 BYTES, FOR A TOTAL OF 24 BITS. THE NUMBERS
284 *      ARE STORED IN TWO'S COMPLEMENT NOTATION. THE HIGH ORDER
285 *      BIT IS THE SIGN BIT, THE NEXT BIT (100Q) IS THE MOST SIGNIFICANT
286 *      DATA BIT. NOTE THAT FLOATING POINT NUMBERS SHOULD ALWAYS
287 *      BE NORMALIZED, SO THAT THE MOST SIGNIFICANT DATA BIT IS THE
288 *      OPPOSITE OF THE SIGN BIT'S VALUE.
289 *
290 *      THE FLOATING POINT ROUTINES SUPPLIED WILL NOT OPERATE ON NON-
291 *      NORMALIZED DATA.
292 *
293 *      EXAMPLES:
294 *      -----
295 *
296 *      DECIMAL NUMBER      M3 M2 M1 EX
297 *
298 *      1.0                  000 000 100 201
299 *      0.5                  000 000 100 200
300 *      0.25                000 000 100 177
301 *      10.0                000 000 120 204
302 *      .0                  000 000 000 000
303 *      0.1                 146 146 146 175
304 *      -1.0                000 000 200 200
305 *      -0.5                000 000 200 177
306 *      -10.0               000 000 260 204
  
```

BASIC - WINTER BASIC INTERPRETER,
 FLOATING POINT ROUTINES.

HEATH X8ASH V1.0 02/18/77
 13:12:46 01-APR-77 PAGE 7

```

309 **      FPADD - FLOATING POINT ADD.
310 *
311 *      ACCX = ACCX + (DE)
312 *
313 *      ENTRY  (DE) = POINTER TO 4 BYTE FP VALUE
314 *      EXIT   ACCX = RESULT
315 *      SUPPLIED VALUE UNCHANGED
316 *      USES   A,F
317
318
319 FPADD EQU *
```

```

321 **      FPSUB - FLOATING POINT SUBTRACT.
322 *
323 *      FPSUB COMPUTES (DE) - ACCX
324 *
325 *      ENTRY  (DE) = POINTER TO 4 BYTE FP VALUE
326 *      EXIT   ACCX = RESULT
327 *      SUPPLIED VALUE UNCHANGED
328 *      USES   A,F
329
330
331 FPSUB EQU *
```

```

333 **      FPNRM - FLOATING POINT NORMALIZE.
334 *
335 *      FPNRM NORMALIZES THE CONTENTS OF (ACCX).
336 *
337 *      ENTRY  NONE
338 *      EXIT   (ACCX) NORMALIZED
339 *      USES   A,F
340
341
342 FPNRM EQU *
```

```

344 **      FPNEG - FLOATING POINT NEGATE.
345 *
346 *      FPNEG NEGATES THE CONTENTS OF ACCX.
347 *
348 *      ENTRY  NONE
349 *      EXIT   (ACCX) = -(ACCX)
350 *      USES   A,F
351
352
353 FPNEG EQU *
```

BASIC - WINTER BASIC INTERPRETER,
FLOATING POINT ROUTINES.

HEATH X8ASM V1.0 02/18/77
13:12:47 01-APR-77 PAGE 8

355 ** FPMUL - FLOATING POINT MULTIPLY.
356 *
357 * ENTRY (DE) = ADDRESS OF Y
358 * EXIT ACCX = ACCX * Y
359 * USES A,F
360
361
362 FPMUL EQU *

364 ** FPDIV - FLOATING POINT DIVIDE.
365 *
366 * ACCX = ACCX/Y
367 *
368 * ENTRY (DE) = POINTER TO Y
369 * EXIT (ACCX) = RESULT
370 * USES A,F
371
372
373 FPDIV EQU *

BASIC - WINTER BASIC INTERPRETER.
 ASCII/FLOATING CONVERSION ROUTINES.

HEATH XBASM V1.0 02/18/77
 13:12:48 01-APR-77 PAGE 9

```

376 **      ATF - ASCII TO FLOATING.
377 *
378 *      ATF CONVERTS AN ASCII STRING INTO A FLOATING POINT VALUE
379 *      IN ACCX.
380 *
381 *      SYNTAX
382 *
383 *      NNNN [ ,NNN] [E [+ -] NN]
384 *
385 *      ENTRY  (HL) = ADDRESS OF TEXT
386 *      EXIT   (HL) UPDATED
387 *             (ACCX) = VALUE
388 *      USES   A,F,H,L
389 *
390
---*---
391 ATF      EQU      *
```

```

393 **      FTA - FLOATING TO ASCII.
394 *
395 *      FTA CONVERTS A FLOATING POINT NUMBER INTO AN ASCII
396 *      REPRESENTATION.
397 *
398 *      ENTRY  (ACCX) = VALUE
399 *             (HL) = ADDRESS TO STORE TEXT
400 *      EXIT   (A) = LENGTH OF STRING DECODED
401 *             (DE) = ADDRESS OF LAST BYTE
402 *      USES   A,F,D,E
403 *
404
---*---
405 FTA      EQU      *
```



BASIC - WINTER BASIC INTERPRETER.

HEATH X8ASM V1.0 02/18/77

FLOATING POINT CONSTANTS.

13:12:48 03-APR-77 PAGE 10

408 ** FLOATING POINT VALUES.

409 *

410

411 LON G

LIST GENERATED BYTES

412

---*--- 000 000 100 413 FP1.0 DB 0.0,100Q,201Q

201

---*--- 000 000 120 414 FP10. DB 0.0,120Q,204Q

204

---*--- 146 146 146 415 FP0.1 DB 146Q,146Q,146Q,175Q

175

---*--- 000 000 000 416 FP0.0 DB 0.0,0.0

000

---*--- 022 170 233 417 NPI.2 DB 022Q,170Q,233Q,201Q -PI/2

201

---*--- 022 170 233 418 NPI2 DB 022Q,170Q,233Q,203Q -PI*2

203

---*--- 022 170 233 419 NPI DB 022Q,170Q,233Q,202Q -PI

202

---*--- 022 170 233 420 NPI.4 DB 022Q,170Q,233Q,200Q -PI/4

200

---*--- 356 207 144 421 PI.4 DB 356Q,207Q,144Q,200Q PI/4

200

422

423

---*--- 425 END

ASSEMBLY COMPLETE

425 STATEMENTS

0 ERRORS

20312 BYTES FREE



APPENDIX D

Entry Points to BASIC Utility Routines

ADDRESSES FOR

BENTON HARBOR BASIC
ISSUE # 05.01.xx

ACCX	040.066	ACCY	040.074
ATF	065.144	CVX	060.171
CXV	060.211	CXY	060.206
ERR.CC	057.100	ERR.DO	057.110
ERR.DE	057.105	ERR.IC	057.154
ERR.IN	057.113	ERR.IU	057.116
ERR.ND	057.151	ERR.NV	057.121
ERR.OV	057.124	ERR.RE	057.127
ERR.SC	057.146	ERR.SN	057.132
ERR.SR	057.143	ERR.SY	057.135
ERR.TO	057.140	ETT.TP	057.162
ERR.UD	057.157	FPO.1	072.101
FP1.0	072.071	FP10.	072.075
FPADD	063.033	FPDIV	064.060
FPMUL	063.274	FPNEG	063.260
FPNRM	063.207	FPSUB	063.173
FTA	066.006	IFIX	060.373
IFLT	061.031	NPI.2	072.105
NP12	072.111	TDI	062.327
USRFCN	072.065	XCY	062.346
ZRO	063.000		



INDEX

NOTE: Numbers printed in a bold type face refer to examples of the indicated statement or function.

- Absolute Value, 5-55
- Addition, 5-12, 5-14
- AND, 5-17
- Arc Tangent Function, 5-56
- Arithmetic, 5-7
- Arithmetic, Functions, 5-55 ff,
- Arithmetic Operators, 5-12
- Arithmetic Priority, 5-12
- Arrays, 5-10 ff, 5-21, 5-29, 5-30
- Assignment Statement, 5-9, 5-38
- Asterisk, 5-6, 5-12

- Backspace, changing of, 5-61 (0-20)
- BASIC File, 0-15 ff, 5-26
- Basic Statements, 5-21
- Blanks (spaces), 5-63
- Boolean Values, 5-8
- Brackets, 5-22

- CTRL-H, 5-61
- CTRL-Q, 5-62
- CTRL-S, 5-62
- Checksum Error, 5-26, 5-28
- CLEAR, 5-29, 5-49
- Colon, 5-21, **5-30**, 5-37
- Comma, 5-43 ff,
- Command Completion, 5-6, 5-62
- Command Mode, 5-19 ff, 5-29
- Comments, 5-48
- Continue, 5-19, 5-23, 5-53, 5-51, 5-61
- Control-C, 5-61
 - Abort List, 5-40
- Control-O, 5-40
 - Abort List, 5-40
- Cosine Function, 5-56

- DATA, 5-47 ff, 5-49,
- Data Exhausted, 5-67
- Data Only Statement,
 - One Line, 5-48
- Decimal Notation, 5-8
- DEF FN, 5-51
- DIM (Dimension), 5-10, 5-11, 5-30
- Discard Flag CTRL-O, 5-62
- Discard Flag CTRL-P, 5-62
- Divide by Zero, 5-65
- Division, 5-12
- Double Commas, 5-45 ff,
- DUMP, 5-24

- END, 5-51
- Equal Sign, 5-16, 5-22, 5-39
- Errors, 5-65 ff, 5-42
- Errors Recover, 5-65
- ERROR Table, 5-67
- Exponential Format, 5-7
- Exponential Function, 5-56
- Exponential Notation, 5-7
- Exponentiation, 5-12 ff,
- Expressions, 5-12

- False, 5-16
- FOR, **5-20**, **5-30**, **5-33**
- Free Space Function (FRE), 5-60
- Functions, Predefined, 5-55 ff,

- GOSUB, 5-34, **5-35**, 5-35 ff,
- GOTO, 5-37

- High Memory, 5-5

- iexp, 5-22
- IF GOTO, 5-38
- IF THEN, 5-16, 5-38
- IGNORE, 5-26, 5-28
- Immediate Execution, 5-19
- INPUT, 5-52
- Input and Line Input, 5-59
- Inputting Control, 5-61
- Integer Function, 5-56
- Integer Numbers, 5-7
- I/O MAP, 0-49

- Label File, 0-12 ff,
- LET, 5-39
- Lexical Rules, 5-63
- Line Deletion, 5-64
- Line Insertion, 5-64
- Line Length, 5-64
- Line Numbers, 5-21
- Line Replacement, 5-64
- LIST, 5-40, **5-63**
- LOAD, 5-24
- Loading Basic, 5-6
- Logarithm Function, 5-56
- Loop, 5-33 ff,

- Map, I/O, 0-49
- Map, MEMORY, 0-50
- Memory, 5-5
 - Map, 0-50
- Multiple Statements, 5-20
- Multiplication, 5-12 ff,

- “Name”, 5-22
- Negation, 5-12, 5-13
- Nesting Depth, 5-34, 5-37
- Nesting, 5-34 ff
- nexp, 5-22 ff
- NEXT, **5-20, 5-30, 5-33**, 5-30 ff
- NOT, 5-12, 5-17
- Numeric Data, 5-7
- NXT, 5-50

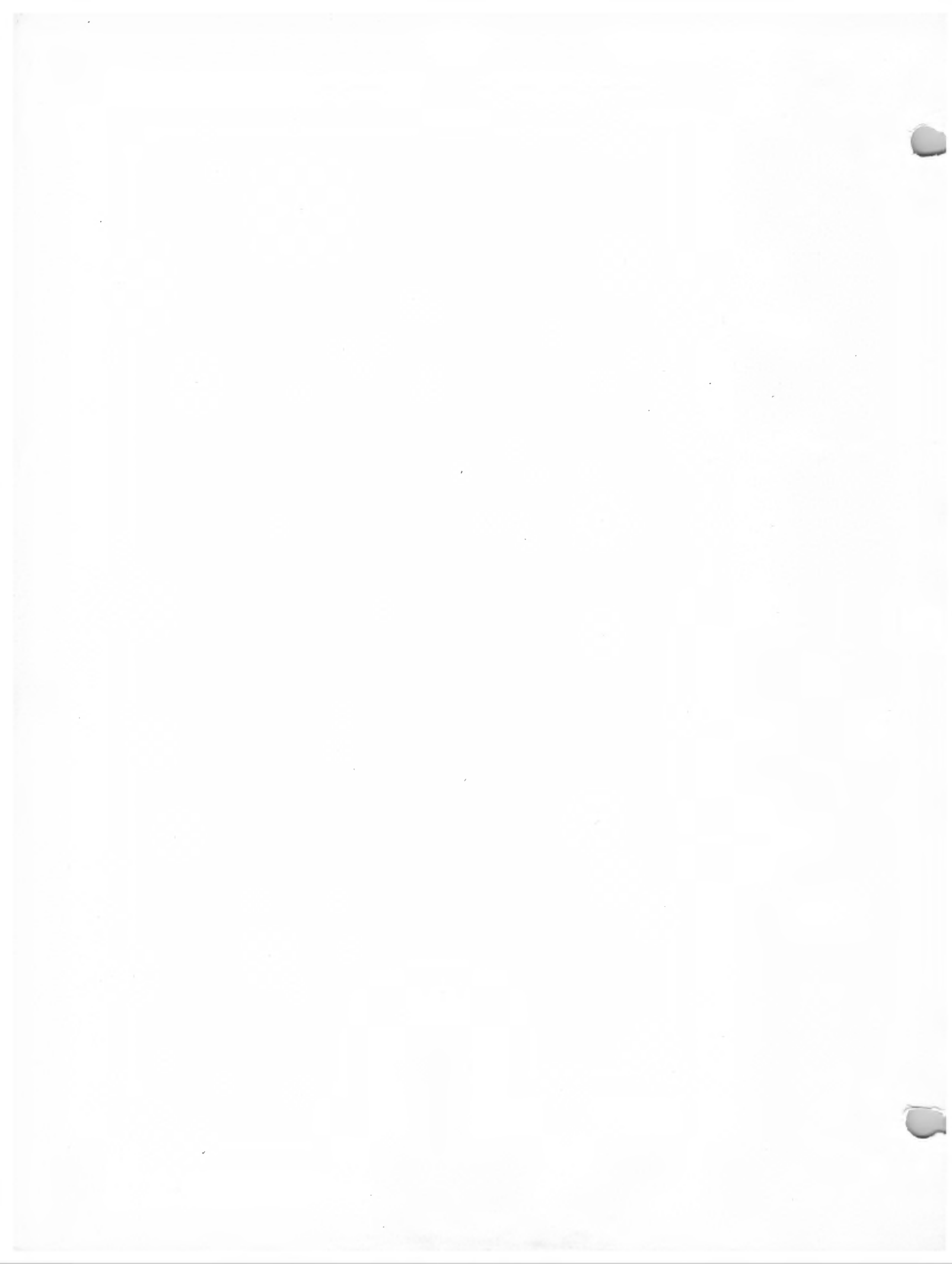
- ON . . . GOSUB, 5-40
- ON . . . GOTO, 5-40
- Operators, 5-12
- OR, 5-17
- OUT, 5-40
- Output Port, 5-40
- Output Restoration, 5-62
- Output Suspension, 5-62
- Outputting Control, 5-62

- PAD Function, 5-57
- Parentheses, 5-13
- PAUSE, 5-41
- PEEK, 5-57
- POKE, 5-41
- PORT, 5-42
- Position Function, 5-57
- Predefined Functions, 5-55 ff,
- PRINT, **5-31, 5-32**, 5-43 ff
- Priority, Arithmetic, 5-12 ff,
- Program Loop, 5-30
- Program Only Mode, 5-21 ff, 5-29, 5-51
- Prompt,
 - Basic, 5-6, 5-61
 - Input, 5-52

- Quotes, Input, 5-52
 - Print, 5-43
 - Load, 5-25
 - Dump, 5-24

- Random Function, 5-57
- READ, 5-47 ff,
- Real Numbers, 5-7
- Record, 0-12 ff,
- Relational Operators, 5-16
- REM (Remark), **5-48**
- RESTORE, 5-49
- RETURN, 5-35 ff,
- RND, 5-57
- Rubout, changing of, 5-61 (0-20)
- RUN, 5-26

SCRATCH, 5-27
Semicolon, 5-45 ff,
Sep, 5-22
Sequence Error, 5-24, 5-28
SGN, 5-59
Sign Function, 5-59
Sine Function, 5-59
Single Statements, 5-20
Single Step Execution, 5-50
Space Function, 5-60
Spaces, see "Blanks", 5-63
Special Feature Functions, 5-55 ff,
SQUARE (Example), 5-20, **5-49**
Square Root Function, 5-59
Statement Length, 5-21
Statements, 5-20 ff,
Statement Types, 5-21
Step, FOR/NEXT, 5-30 ff,
STEP, 5-50
STOP, 5-53
Subroutines, 5-35 ff,
Subscripted Variables, 5-10
Subtraction, 5-12, 5-14
SURE, 5-27
TAB Function, 5-60
Tape Error, 5-26
Text Rules, 5-63
Trailing Blanks, 5-20
True, 5-16
Truncation, 5-8
Unary Operators, 5-12 ff
USE Error, 5-22
Var, 5-22
Variables, 5-9, 5-29
Verify, 5-27



APPENDIX C

BENTON HARBOR BASIC

and

**EXTENDED BENTON HARBOR
BASIC**

This Appendix is a typical “User’s Manual”, as supplied to describe the features of a particular version of BASIC.

This BENTON HARBOR BASIC User’s Manual illustrates the scope of the special instructions you can expect from such a user’s Manual, and will further illustrate the features of BENTON HARBOR BASIC, which we have used for our course material.

TABLE OF CONTENTS

INTRODUCTION

Manual Scope	17-6
Hardware Requirements	17-6
Loading and Running BASIC	17-7
Benton Harbor BASIC and Extended Benton Harbor BASIC	17-7
Command Completion	17-7

BASIC ARITHMETIC

Data Types	17-9
Variables	17-11
Subscripted Variables	17-12
Expressions	17-14
Arithmetic Operators	17-14
Relational Operators	17-18
Boolean Operators	17-19

STRING MANIPULATION

String Variables	17-21
String Operators	17-22

THE COMMAND MODE

Using The Command Mode For Statement Execution	17-23
--	-------

BASIC STATEMENTS

Line Numbers	17-25
Statement Types	17-25
Command Mode Statements	17-27
Statements Valid In the Command or Program Mode	17-33
Program Mode Statements	17-58

PREDEFINED FUNCTIONS

Introduction	17-61
Arithmetic and Special Feature Functions	17-61
STRING Functions (Extended BASIC only)	17-68

EDITING COMMANDS

Control-C, CNTRL-C	17-71
Inputting Control	17-71
Outputting Control	17-72
Command Completion	17-72
Enforced Lexical Rules	17-73
General Text Rules	17-73

ERRORS

Error Messages	17-75
Recovering from Errors	17-75

BASIC ERROR TABLE	17-77
-------------------------	-------

APPENDIX A

Loading from the Software Distribution Tape	17-81
Loading from a Configured Tape	17-82

APPENDIX B

Numeric Data	17-83
Boolean Data	17-83
String Data (Extended Basic Only)	17-83
Variables	17-83
Sunscripted Variables	17-84
Arithmetic Operators	17-84
Relational Operators	17-84
Boolean Operators	17-85
String Variables	17-85
String Operators	17-85
Line Numbers	17-85
The Command Mode	17-85
Multiple Statements on One Line	17-85
Command Mode Statements	17-86
Command and Program Mode Statements	17-87
Program Mode Statements	17-90
Predefined Functions	17-91
Editing Commands	17-93

APPENDIX C

BASIC Utility Routines	17-95
------------------------------	-------

APPENDIX D

Entry Points to Utility Routines	17-107
--	--------

APPENDIX E

An Example of USR	7-111
-------------------------	-------

INDEX	17-113
-------------	--------

TAB GUIDE

BASIC ARITHMETIC	
STRING MANIPULATION	
THE COMMAND MODE	
BASIC STATEMENTS	
PREDEFINED FUNCTIONS	
EDITING COMMANDS	
ERRORS	
BASIC ERROR TABLE	
APPENDIX A	
APPENDIX B	
APPENDIX C	
APPENDIX D	
APPENDIX E	

INTRODUCTION

BENTON HARBOR BASIC is a conversational programming language which is an adaptation of Dartmouth BASIC*. (BASIC is an acronym for Beginners' All Purpose Symbolic Instruction Code.) It uses simple English statements and familiar algebraic equations to perform an operation or a series of operations to solve a problem. BENTON HARBOR BASIC is an interpretive language, compact enough to run in a Heath H8 computer with minimal memory, yet powerful enough to satisfy most problem-solving requirements. The interpretive structure of BASIC affords excellent facilities for the detection and correction of programming errors. It uses advanced techniques to perform intricate manipulations and to express problems more efficiently.

Two versions of BENTON HARBOR BASIC are available. Extended BENTON HARBOR BASIC (EX. B.H. BASIC) with strings provides character string manipulation and advanced functions. BENTON HARBOR BASIC (B.H. BASIC) does not have strings and some advanced functions, and so uses less memory. The user may operate B.H. BASIC in an H8 computer with 8K of memory.

Manual Scope

This Manual is written for the user who is already familiar with the language BASIC. It also describes the extended implementation of Dartmouth BASIC and, in so doing, provides a brief summary of the language. However, this manual is not intended as an instruction Manual for the language BASIC. If you are not familiar with BASIC, we suggest that you obtain the Heathkit Continuing Education course entitled "Basic Programming," Model EC-1100, before attempting to use this Manual.

Hardware Requirements

Extended BENTON HARBOR BASIC (with strings) runs on an H8 computer with a minimum of 12K bytes of random access memory. BENTON HARBOR BASIC (without strings) runs on an H8 with a minimum of 8K memory. Both versions require a console terminal, its appropriate interface card, and a mass storage device such as a cassette or paper tape reader/punch.

Both BASICs automatically measure the maximum amount of unbroken memory above the starting point at 8K (40,100 offset octal). They use all available memory unless the high memory limit is configured otherwise during the system configuration procedure (see "Product Installation" on Page 0-19 in the "Introduction" to this Software Reference Manual).

*BASIC is a registered trademark of the Trustees of Dartmouth College.

Benton Harbor BASIC and Extended Benton Harbor BASIC

This Manual covers both BENTON HARBOR BASIC (B.H. BASIC) and Extended BENTON HARBOR BASIC (EX. B.H. BASIC). Information that applies only to Extended BASIC is printed in a different type face, as shown below. For example:

Strings are only used in EX. B.H. BASIC, except in PRINT statements.

Anything not marked in this different type face applies to BASIC. References to "BASIC" apply to both BENTON HARBOR BASIC and to Extended BENTON HARBOR BASIC. BASIC is summarized in "Appendix B."

Loading and Running BASIC

BASIC is distributed in binary load format on cassette tapes or paper tape. It is loaded in accordance with the software configuration guide, outlined in "Product Installation" on Page 0-19. A condensed version of this loading procedure is given in "Appendix A" of this Manual. Once a system BASIC tape is configured, you can load the configured tape, using the internal PAM-8 loader, and start it by pressing the GO key. EX. B.H. BASIC and B.H. BASIC use the asterisk (*) as a prompt character.

Command Completion

Both the B.H. BASIC and EX. B.H. BASIC employ command completion. BASIC examines each character as you type it on the console keyboard, and when sufficient information is received to uniquely identify one particular command, BASIC finishes typing the command for you. For example, once the letters PR have been typed, the command PRINT is uniquely defined. Therefore, BASIC supplies letters INT and the required blank following the T.

BASIC also watches your spelling. As commands are being typed, letter combinations leading to non-existent commands are not accepted and the console terminal bell is rung.

BASIC ARITHMETIC

Data Types

BASIC supports three different data types:

1. Numeric data.
2. Boolean data.
3. String data.

NUMERIC DATA

BASIC accepts real and integer numbers. A real number contains a decimal point. BASIC assumes a decimal point **after** integer data. Any number can be used in mathematical expression without regard to its type. Real numbers must be in the approximate range of 10^{-38} to 10^{+37} . Integer numbers must lie in the range of 0 to 65535. All numbers used in BASIC are internally represented in floating point, which allows approximately 6.9 digits of accuracy. Numbers may be either negative or positive.

In addition to integer and real numbers, BASIC recognizes a third format. This format, called exponential notation, expresses a number as a decimal number raised to a power of 10. The exponential form is

$$XXE(\pm)NN$$

where E represents the algebraic statement “times ten to the power of,” XX represents up to a six digit integer or real number, and NN represents an integer from 0 to 38. Thus, the number is read as “XX times 10 to the \pm power of NN.”

Numeric data in all three forms may be used in the immediate mode, program mode in data statements, or in response to READ and INPUT statements.

Unless otherwise specified, all the numbers including exponents are presumed to be positive.

The results of BASIC computations are printed as decimal numbers if they lie in the range of 0.1 to 999999*. If the results do not fall in this range, the exponential format is used. BASIC automatically suppresses all leading and trailing zeros in real and integer numbers. When the output is in exponential format, it is in the form

(±) X . XXXXXE (±) NN

The following are examples of typical inputs and the corresponding output. Note the dropping of leading and trailing zeros, truncation to six places of accuracy, conversion to exponential notation when necessary, and conversion to decimal notation where permitted.

<u>INPUT NUMBER</u>	<u>OUTPUT NUMBER</u>	<u>COMMENTS</u>
0.1	.1	(leading zero dropped)
.0079	7.90000E-03	(<.1 converts to exponential)
0022	22	(leading zeros dropped)
22.0200	22.02	(trailing zeros dropped)
999999	999999	(format maintained)
1000000	1.00000E+06	(converted to exponential)
100000007	1.00000E+08	(truncated to 6 places)
-10.1E+2	-1010	(converted to decimal format)

BOOLEAN VALUES

Boolean values are a subclass of numeric values. Values representing the positive integers from 0-65,535 (2^{16-1}) may be used as Boolean data. When using numeric data as Boolean values, the numeric data represents the equivalent 16-bit binary numbers. Fractional parts of numeric data used with Boolean operators are discarded. If the numeric value with the fractional part does not fall into the range of 0-65,535, an illegal number error is generated.

STRING DATA (Extended BASIC Only)

Extended BASIC handles data in a character string format. Data elements of this type are made up of a string of ASCII characters up to 255 characters in length. Extended BASIC provides operators and functions to manipulate string data. String values in either programmed text or data must always be enclosed by quotation marks (""). Any printable ASCII character (with the exception of the quotation mark itself) may appear in an Extended BASIC string. In addition to the printable ASCII characters, the line feed and bell characters are also permitted. A string may not be typed on more than one line. A carriage return is rejected as an illegal string character.

*NOTE: This may be changed in EX. B.H. BASIC. See "CNTRL 1," Page 17-35.

Variables

A BASIC variable is an algebraic symbol representing a number. Variable naming adheres to the Dartmouth specification. That is, variable names consist of one alphabetic character which may be followed by one digit (zero to nine). The following is a list of acceptable and unacceptable variables, and the reason why the variable is unacceptable.

<u>ACCEPTABLE VARIABLES</u>	<u>UNACCEPTABLE VARIABLES</u>	<u>REASON FOR UNACCEPTABILITY</u>
C	2C	A digit cannot begin a variable.
A5	AF	A second character in a variable must be a number (0-9).
D	3	A single number is not an acceptable variable.
L2	\$2	The first character of a variable must be a letter (A-Z).

Subscripted variables, string variables, and subscripted string variables are permitted. See "Subscripted Variables," Page 17-12, and "String Manipulation" on Page 17-21.

A value is assigned to a variable when you indicate the value in a LET, READ, or INPUT statement. These operations are discussed in "LET" (Page 17-46), "PRINT" (17-50), and "INPUT AND LINE INPUT" (Page 17-59).

The value assigned to a variable changes each time a statement equates the variable to a new value. The RUN command sets all variables to zero (0). Therefore, it is only necessary to assign an exact value to a variable when an initial value other than zero is required.

Subscripted Variables

In addition to the variables described above, BASIC permits subscripted variables. Subscripted variables are of the form:

$$A_n (N_1, \dots, N_8),$$

where A is the variable letter, n is a number (optional) 0-9, and N_1 thru N_8 are the integer dimensions of the variable. Subscripted variables provide you with the ability to manipulate lists, tables, matrices, or any set of variables. Variables are allowed one to eight subscripts.

The use of subscripts permits you to create multi-dimensional arrays of numeric and string variables. It is important to note that a dimensioned variable is distinguished from a scalar value of the same name. For example, all four of the following are distinct variables:

$$A, A(N), A\$*, A\$(N)*$$

When you are referencing a subscripted variable, each element in the subscript list may consist of an arbitrarily complex expression so long as it evaluates to a numeric value within the allowable range for the indicated dimension. Thus, the subscripted variable A(5,5), would be dimensioned as:

$X = A(2, 3)$	is legal
$X = A(2+2, VAL("4.0"))$	is legal as it is equivalent to A(4,4)
$X = A(2, "4.0")$	is not legal as ("4.0" is a string)

*NOTE: The \$ indicates a string variable valid only in EX. B.H. BASIC. See Section 3.

The following are graphic illustrations of simple subscripted variables. In these particular examples, a simple variable (A) is followed by one or two integer expressions in parentheses. For example,

A(I)

where I may assume the values of 0 to 5, allows reference to each of the six elements A(0), A(1), A(2), A(3), A(4), and A(5). A graphic representation of this 6-element, single-dimension array is shown below. Each box represents a memory location reserved for the value of the variable of the indicated name. Often, the entire array is referred to as A(.

A(0)
A(1)
A(2)
A(3)
A(4)
A(5)

NOTE: Subscripted variables begin at zero. Therefore, the previous example 0 to 5 defines six elements.

A two dimensional array B(I, J) allows you to refer to each of the 20 elements B(0,0), B(0,1), B(0,2), . . . , B(0,J), . . . , B(I,J).

This is graphically illustrated as follows, for B(3,4).

J					
I	B(0,0)	B(0,1)	B(0,2)	B(0,3)	B(0,4)
	B(1,0)	B(1,1)	B(1,2)	B(1,3)	B(1,4)
	B(2,0)	B(2,1)	B(2,2)	B(2,3)	B(2,4)
	B(3,0)	B(3,1)	B(3,2)	B(3,3)	B(3,4)

NOTE: A variable cannot be dimensioned twice in the same program. *In EX. B.H. BASIC, a clear may be used to destroy an array, allowing you to reuse it.*

BASIC does not presume any dimension. Therefore, the DIMension (DIM) statement must be used to define the maximum number of elements in any array. It is described in “DIM (DIMENSION)” on Page 17-36.

Expressions

An expression is a group of symbols to be evaluated by BASIC. Expressions are composed of numeric data, Boolean data, string data, variables, or functions. In an expression, these are alone or combined by arithmetic, relational, or Boolean operators.

The following examples show some expressions BASIC recognizes.

ARITHMETIC EXPRESSIONS	BOOLEAN EXPRESSIONS	STRING EXPRESSIONS	DESCRIPTION
1.02	255	“YES”	Data
1.02 + 16	255 OR 003	“YES” + “NO”	Combined
A < B		“YES” < “NO”	Relational

A major feature of BASIC is its extensive use of expressions in situations when many other BASICs only permit variables or numbers. This feature permits you to perform very sophisticated operations within a particular command or function. It is important to note that not all expressions can be used in all statements. The explanations describing the individual statements detail any limitations.

Arithmetic Operators

BASIC performs *exponentiation* (in EX. B.H. BASIC only), multiplication, division, addition, and subtraction. BASIC also supports two unary operators (– and NOT). The asterisk (*) is used to signify multiplication and the slash (/) is used to indicate division. *Exponentiation is indicated by the up arrow (↑).*

THE PRIORITY OF ARITHMETIC OPERATIONS

When multiple operations are to be performed in a single expression, an order of priority is observed. The following list shows the arithmetic operators in order of descending precedence. Operators appearing on the same line are of equal precedence.

–(Unary)	(negation)
↑	(<i>exponentiation</i>)
* /	(multiplication division)
+ –	(addition subtraction)

Parentheses are used to change the precedence of any arithmetic operations, as they are in common algebra. Parentheses receive top priority. Any expression within parentheses is evaluated before an expression without parentheses. The innermost leftmost parenthetical expression has the greatest priority.

UNARY OPERATORS

BASIC supports two unary operators: $-$ and NOT. These operators are referred to as unary because they require only one operand. For example:

```
A = -2
C = NOT D
```

The unary operator ($-$) performs arithmetic negation. The NOT operator performs Boolean negation. See Page 17-19.

EXPONENTIATION (EXTENDED BASIC ONLY)

Exponentiation (\uparrow) is used to raise numeric or variable data to a power. For example:

*$A = B \uparrow 2$ is equivalent to $A = B * B$.*

*NOTE: The operand must not be negative. The exponent may be negative. A negative operand generates a syntax error. For greatest efficiency, $B \uparrow 2$ should be written as $B * B$ and $B \uparrow 3$ should be written as $B * B * B$. All other powers should use the \uparrow .*

MULTIPLICATION AND DIVISION

BASIC uses the asterisk ($*$) and the slash ($/$) as symbols to perform the algebraic operations of multiplication and division respectively. Both multiplication and division require numeric data as operands.

The following examples use the multiplication and division operators:

```
*PRINT 2*6
12

*PRINT 6/3
2

*PRINT 6/3*2
4

*
```

NOTE: This last expression evaluates to 4, not 1; as $*$ and $/$ have equal precedence and therefore the leftmost operator is evaluated first.

ADDITION AND SUBTRACTION

The plus sign (+) and the minus sign (−) perform arithmetic addition and subtraction. *In addition, the plus operator (+) performs string concatenation if both operands are string data. Concatenation is restricted to Extended BASIC.* The following examples use the plus and minus operators:

```
*PRINT 3
3
*PRINT 3+5
8

*PRINT 10-3
7

*PRINT "HEATH" + " " + "H8"
HEATH H8
* }
```

extended BASIC only

SUMMARY

In any given expression, BASIC performs arithmetic operations in the following order:

1. Parentheses have top priority. Any expression in parentheses is evaluated prior to a nonparenthetical expression.
2. Without parentheses, the order of priority is:
 - a. Unary minus and NOT (equal priority).
 - b. *Exponentiation (proceeds from left to right).*
 - c. Multiplication and division (equal priority, proceeds from left to right).
 - d. Addition and subtraction (equal priority, proceeds from left to right).
3. If the rules in either 1 or 2 do not clearly designate the order of priority, the evaluation of expression proceeds from left to right.

The following examples illustrate these principles. *The expression $2 \uparrow 3 \uparrow 2$ is evaluated from left to right:*

1. $2 \uparrow 3 = 8$ (*left-most exponentiation has highest priority*).
2. $8 \uparrow 2 = 64$ (*answer*).

The expression $12/6*4$ is evaluated from left to right since multiplication and division are of equal priority:

1. $12/6 = 2$ (division is the left-most operator).
2. $2*4 = 8$ (answer).

The expression $6+4*3\uparrow 2$ evaluates as:

1. $3\uparrow 2 = 9$ (*exponentiation has highest priority*).
2. $9*4 = 36$ (multiplication has second priority).
3. $36+6 = 42$ (addition has lowest priority; answer).

Parentheses may be nested, (enclosed by additional **sets** of parentheses). The expression in the innermost set of parentheses is evaluated first. The next innermost left justified is second, and so on, until all parenthetical expressions are evaluated. For example:

$$6 * ((2\uparrow 3 + 4) / 3)$$

Evaluates as:

1. $2\uparrow 3 = 8$ (*exponentiation in parentheses has highest priority*).
2. $8+4 = 12$ (addition in parentheses has next highest priority).
3. $12/3 = 4$ (next innermost parentheses are evaluated).
4. $4*6 = 24$ (multiplication outside of parentheses is lowest priority).

Parentheses prevent confusion or doubt when you are evaluating the expression. For example, the two expressions

$$D * E \uparrow 2 / 4 + E / C * A \uparrow 2$$

$$((D * (E \uparrow 2)) / 4) + ((E / C) * (A \uparrow 2))$$

are executed identically. However, the second is much easier to understand.

Blanks should be used in a similar manner, as BASIC ignores blanks (except when they are part of a string enclosed in quotation marks). The two statements:

```
10 LET B = 3 * 2 + 1
10 LET B=3*2+1
```

are identical. The blanks in the first statement make it easier to read.

Relational Operators

Relational operators compare two variables or expressions. They are generally used with an IF THEN statement. The result of a comparison by the relational operators is either a true or a false. A false is represented by zero, and true is represented by 65535 ($2^{16}-1$). NOTE: These values are chosen so when they are used as Boolean values, false is all zeros and true is all ones.

The following table lists relational operators as used in BASIC.

<u>ALGEBRATIC SYMBOL</u>	<u>BASIC SYMBOL</u>	<u>EXAMPLE</u>	<u>MEANING</u>
=	=	A=B	A is equal to B.
<	<	A<B	A is less than B.
≤	<=	A<=B	A is less than or equal to B.
>	>	A>B	A is greater than B.
>	>=	A>=B	A is greater than or equal to B.
≠	<>	A<>B	A is not equal to B.

The symbols =<, =>, >< are not accepted and BASIC generates a syntax error if they are used.

The following examples show the results of using relational operators.

```
*PRINT 3<4      (true)
65535

*PRINT 4<3      (false)
0
```

EX. B.H. BASIC and B.H. BASIC differ from most other versions in the use of the relational operator. When you are using BASIC, you may use the relational operators in any expression. When the expression is evaluated, the appropriate numeric answer (0 or 65535) will be used as the answer to that expression.

Boolean Operators

OR

The operator OR performs a Boolean OR on the two integer operands. The integer operands (which must lie in the range of 0 to 65535) are converted to 16-bit binary numbers. The Boolean (logical) 16-bit OR is applied and the result is returned to the equivalent integer representation. NOTE: As the Boolean value chosen to represent true (65535) and false (0), the OR operator implements a standard truth table OR function. For example:

```
*PRINT 132 OR 255      00000000 10000100    132
255                    00000000 11111111    255
                        00000000 11111111    255
```

and

```
*PRINT (3>2) OR (4>9)
65535
```

AND

The AND operator performs a Boolean (logical) AND on the two integer operands. These integer operands must lie in the range of 0 to 65535. The integer operands are converted into 16-bit binary numbers and the logical AND is performed. The result is returned to the equivalent integer representation. NOTE: The AND operator implements a standard AND truth table on the values true (65535) AND false (0). For example:

```
*PRINT 132 AND 255      00000000 10000100    132
132                    00000000 11111111    255
*                        00000000 10000100    132
```

and

```
*PRINT (3>2) AND (9>7)
65535
```

NOT

The NOT operator Boolean negation. That is, the numeric value of the variable is converted into a 16-bit Boolean data value; each **bit** is inverted, and the 16-bit binary number is restored to numeric data. For example:

```
*PRINT NOT 0      0 = 00000000 00000000    and
65535      65535 = 11111111 11111111
*
```


STRING MANIPULATION

Extended BENTON HARBOR BASIC is capable of manipulating string information. A string is a sequence of characters treated as a single unit of an expression. It can be composed of alphanumeric and other printing characters. An alphanumeric string contains letters, numbers, blanks, or any combination of these characters. A character string may not exceed 255 characters. The blank, bell, and line feed are considered to be printing characters.

String Variables

The dollar sign (\$) following a variable name indicates a string variable. For example:

B\$
and
L6\$

are string variables. A string variable (B\$) is used in the following example.

```
*B$ = "HI": PRINT B$  
  
HI
```

NOTE: The string variable B\$ is separate and distinct from the variable B.

Any array name followed by the \$ character notes that the dimensioned variable is a string. For example:

L\$(n)	A2\$(n)	(single-dimensional string variables).
D\$(m,n)	H1\$(m,n)	(multiple-dimensional string variables).

The numbers in parentheses indicate the location within the array. See "Subscripted Variables," Page 17-12.

The same variable can be used as a numeric variable and as a string variable in one program. For example, each of the following is a different variable:

B	B(n)
B\$	B\$(m,n)

The following are illegal, as they are double declarations of the same variable.

A\$(n)	A\$(n,m)
--------	----------

String arrays are defined with a dimension (DIM) statement in the same way numerical arrays are defined.

String Operators

Extended BASIC provides you with the ability to manipulate strings. The string manipulation operators are: plus (+), for concatenation, and the relational operators.

CONCATENATION

Concatenation connects one string to another without any intervening characters. This is specified by using the plus (+) symbol and only works with strings. The maximum range of a concatenated string is 255 characters. For example:

```
*PRINT "THE HEATH" + " H8 COMPUTER"
THE HEATH H8 COMPUTER
```

RELATIONAL OPERATORS FOR STRINGS

Relational operators, when applied to strings, indicate alphabetic sequence. The relational comparison is done on the basis of the ASCII value associated with each character, on a character-by-character basis, using the ASCII collating sequence. A null character (indicating that the string is exhausted) is considered to head the collating sequence. For example:

```
*PRINT "ABC" < "DEF"
65536      (The relation shown is true)
*PRINT "ABC">"ABCD"
0          (The relation is false. "ABC" is less than "ABCD.")
```

NOTE: *In any string comparison, trailing blanks are not ignored. For example:*

```
*PRINT "CDE" = "CDE "
0          (The equality is false.)
```

The following table indicates how relational operators are used with string variables in Extended BASIC.

OPERATOR	EXAMPLE	MEANING
=	A\$ = B\$	String A\$ and B\$ are alphabetically equal.
<	A\$ < B\$	String A\$ is alphabetically less than B\$.
>	A\$ > B\$	String A\$ is alphabetically greater than B\$.
<=	A\$ <= B\$	String A\$ is equal to or less than B\$.
>=	A\$ >= B\$	String A\$ is equal to or greater than B\$.
<>	A\$ <> B\$	String A\$ and B\$ are not alphabetically equal.

THE COMMAND MODE

Using The Command Mode For Statement Execution

You may solve a problem in BASIC by using a complete program or by use of the **command** mode. **Command** mode makes BASIC an extremely powerful calculator.

Lines of program material entered for later execution are identified by line numbers. BASIC identifies those lines entered for immediate execution by the absence of the line number. That is to say, statements that begin with line numbers are stored, and statements without line numbers are executed immediately when a carriage return is received. For example*:

```
10 PRINT "THIS IS AN H8 COMPUTER"
```

is not executed when it is entered at the console terminal. However, the statement:

```
*PRINT "THIS IS THE HEATH H8 COMPUTER"
```

when the RETURN key is typed, is immediately executed as:

```
THIS IS THE HEATH H8 COMPUTER
```

The **command** mode of operation is useful in program de-bugging and performing simple calculations which do not justify the writing of a complete program.

For example, in order to facilitate program de-bugging, you may place **STOP** statements liberally throughout a program. Once BASIC encounters a **STOP** statement, the program halts. You can examine and change data values using the **command** mode. The statement

```
CONTINUE
```

is used to continue execution of the program. You can also use the **GOSUB** and **IF** commands. Values assigned to variables remain intact using this technique. A **SCRATCH**, **CLEAR**, or another **RUN** command resets these values.

*NOTE: Strings may be used in B.H. BASIC PRINT statement. However, these strings cannot be manipulated in B.H. BASIC.

The ability to place multiple statements on a single line is an advantage in the **command** mode. For example:

```
*B = 2:PRINT B:PRINT B + 1
2
3
*
```

Program loops are allowed in the **command** mode. For example, a table of squares can be produced as follows:

```
*FOR A = 1 TO 10:PRINT A,A * A:NEXT A
1          1
2          4
3          9
4         16
5         25
6         36
7         49
8         64
9         81
10        100
*
```

Some statements cannot be used in the **command** mode. The INPUT statement, for example, is not available in the **command** mode, and its use results in the USE error message. There are certain command functions in the **command** mode which make no sense when used in the **command** mode. Statements available in the **command** mode are covered in “Command Mode Statements” on Page 17-27 and “Statements Valid in the Command or Program Mode” on Page 17-33.

BASIC STATEMENTS

A program is composed of one or more lines or “statements” instructing BASIC to solve a problem. Each program line begins with a line number identifying the line and its statement. The line number indicates the desired order of statement execution. Each statement starts with an English word specifying the operation to be performed. Single statements are terminated with the return key. Multiple statements are separated by a colon (:) with the last statement terminated by a return (a non-printing character). A DATA statement cannot share a line with other statements. (See Page 17-54.)

Line Numbers

An integer number begins each line in a BASIC program. BASIC executes the program statements in numerical sequence, regardless of the input order. Statement numbers must lie in the range of 1 to 65,535. It is good programming practice to number lines in increments of 5 or 10 to allow insertion of forgotten or additional statements when de-bugging the program.

The length of a BASIC statement must not exceed one line. There is no method to continue a statement to a following line. However, multiple statements may be written on a single line. In this situation each statement is separated by a colon. For example:

```
10 PRINT "VALUES", A, A+1 is a single line print statement, whereas  
10 LET A=12: PRINT A, A+1, A+2 is a line containing two statements, LET and PRINT.
```

Virtually all statements can be used anywhere in a multiple statement line. There are, however, a few exceptions. They are noted in the discussion of each statement. NOTE: Only the first statement on a line can have a line number. Program control cannot be transferred to a statement **within** a line, but only to the beginning of a line.

Statement Types

BENTON HARBOR BASIC supports three different types of statements. First, there are statements valid only in the command mode. These statements are used for loading programs, erasing memory, and other such functions directing BASIC's activities. Second, there are statements valid as both commands or within a program. Third, there are statements valid only within a program. These statements may not be used in the command mode. Most statements fall into the second category. This means they can appear within a program or be typed directly in the command mode and immediately executed.

As noted earlier, some statements valid in both modes may not be meaningful in both modes.

BASIC is designed to allow maximum versatility in its structure. Thus, almost everywhere that BASIC requires a number or a string, BASIC allows a numeric or a string **expression**. For example, you can construct a computed GOTO by simply computing a value for a variable, X. The statement

GOTO X

then redirects the program to the computed line number.

The following three sections are organized as command mode statements, command and program mode statements, and program mode statements. They can be found, respectively in: "Command Mode Statements" (below), "Statements Valid in the Command or Program Mode" (Page 17-33), and "Program Only Statements" (Page 17-58).

To simplify some practical descriptions in these sections and those following, the notation below is used to describe allowed expressions:

1. "iexp" indicates an integer expression, an expression lying in the range of 0 to 65535. The fractional part of any integer expression is discarded when the integer is formed.
2. "nexp" indicates a numeric expression. This may be an integer, decimal, or exponential expression with up to 6 decimal places.
3. "sexp" indicates a string expression. String expressions are limited to a maximum of 255 printing ASCII characters. String expressions are limited to *Extended BASIC*.
4. "sep" indicates a separator. Separators such as the comma and the semi-colon are used to delineate certain portions of BASIC statements.
5. "[]" brackets indicate optional portions of a statement, depending on the exact function desired.
6. "var" indicates a variable. This may be a numeric or string variable, depending upon the example.
7. "name" indicates a string used to identify a date, a program, or a language record.

Command Mode Statements

The command mode statements cannot be used within a program. For example, the RUN statement cannot be used within a program to make it self-starting. Any attempt to incorporate one of these statements within a program generates a USE error message.

BUILD

This statement is used to insert or replace many program lines. The form of the BUILD statement is:

```
BUILD iexp1, iexp2
```

When BUILD is executed, the initial line number iexp1 is displayed on the terminal. Any text entered after the new line number is displayed becomes the new line, replacing any pre-existing line. Once the line is completed by a carriage return, the next line number is displayed. NOTE: If a null entry is given (a carriage return typed directly after the line number is displayed), the line whose number is displayed is eliminated if it existed.

BUILD is illustrated in the following example. CONTROL-C terminates BUILD.

```
*BUILD 100,10
100 PRINT "LINE 100"
110 PRINT "LINE 110"
120 PRINT "LINE 120"
130 < CNTRL-C >      (Control-C typed here)
*LIST
100 PRINT "LINE 100"
110 PRINT "LINE 110"
120 PRINT "LINE 120"
*
```

CONTINUE

CONTINUE begins or resumes the execution of a BASIC program. CONTINUE has the unique feature of not affecting any existing variable values, nor does it affect the GOSUB or FOR stack. CONTINUE is normally used to resume execution after an error the program or after a CONTROL-C stops the program. CONTINUE may be used to enter a program or a specific line (in conjunction with a GOTO). CONTINUE is unlike RUN, which resets all variables, stacks, etc.. The form of the CONTINUE statement is:

```
CONTINUE
```

In the following example, CONTINUE starts the program at a specific line number.

```
*GOTO 100
*CONTINUE      (start execution at line 100)
```

CONTINUE is also useful for entering a program with a variable or variables set at particular values. For example:

```
*A = 23.5      (Program continues execution at Line 230
               with variable A set to the value 23.5,
*GOTO 230
*CONTINUE      regardless of previous program effects on A.)
```

DELETE

The DELETE statement is used to remove several lines from the BASIC source program. The form of the DELETE statement is:

```
DELETE iexp1, iexp2,
```

The lines between and including iexp1 and iexp2 are deleted.

A syntax error is flagged if "iexp1" is greater than "iexp2." Normally, DELETE is used to eliminate a number of lines of text. The SCRATCH command is used to eliminate all text. A RETURN typed directly after a line number eliminates that line. This technique is used to eliminate a single line.

DUMP

The DUMP statement saves the current program text on the mass storage media connected to the load/dump port. This is usually paper tape or cassette. The current program is saved; however, no variables are saved. The specific program name is written with the data so the user may reload the program by the specified name. The form of the DUMP statement is:

```
*DUMP "name"
```

Make the tape drive ready before entering the DUMP statement. BASIC starts the drive, writes the data, and stops the drive. The CONTROL-C can be used to abort the DUMP while in progress. However, if a DUMP is aborted, an incomplete file exists on the tape.

The string "name" may be up to 80 ASCII characters. The normal string ASCII characters are permitted. An example of a DUMP is:

```
*DUMP "STARTREK VER 1.0 03/11/77"
```

This statement dumps the program Startrek version 1.0 dated the 11th of March 1977.

LOAD

The LOAD statement discards the current program in memory. A specified program is loaded from the mass storage device connected to the load/dump port. The form of the LOAD statement is:

```
LOAD "name"
```

The string "name" may consist of up to 80 ASCII characters. The normal string ASCII characters are permitted. BASIC scans the mass storage device until it finds a program whose name matches the specified string. Before destroying the stored information, the user is asked "SURE?." A "Y" reply causes LOAD to proceed. Any other response cancels LOAD.

NOTE: If the name on the mass storage device is longer than the specified name, a match on the supplied characters in the string "name" is valid. Thus, a program may be dumped with extra information entered in the name such as program version number and data. The program can then be loaded without it. For example:

```
*DUMP "STARTREK VER 1.0 03/11/77"
```

This program, Startrek version 1.0 dated 11 March 1977, may be loaded by

```
*LOAD "STARTREK"  
SURE? Y
```

A match is found between the first eight characters of the DUMP string "STARTREK" and the eight characters of the load command "STARTREK." If a null is used as the load string, the next program on the tape is loaded. Therefore, the statement

```
*LOAD ""
```

loads the next BASIC program appearing on the mass storage.

Mass storage media should be made ready before LOAD is executed. BASIC starts and stops the mass storage device. CONTROL-C may be used to abort the load part way through. Use a SCRATCH command to clear the results of an aborted load.

During a load, either one of the following two error messages may be generated:

SEQ ERR and
CHKSUM ERR.

A sequence error (SEQ ERR) is generated if the file records are not in sequence. For example, if two consecutive label records are read an error is generated, as a BASIC file consists of a label file followed by a data file. The form of the sequence error is

SEQ ERR

Type a blank after the SEQ ERR message. This will clear the error. The entire file must be reread.

A checksum error (CHKSUM ERR) is generated if the computed CRC for the record in question does not match the CRC included in the record. The form of the checksum error message is

CHKSUM ERR IGNORE?

A Y in response to the question "ignore" aborts the error message and the next consecutive record is read. Do not ignore the checksum error unless there is no other way to recover the data. If a checksum error is flagged, the chances are very good that the data in the designated record is faulty. If you attempt to use such bad data, it may cause BASIC to crash.

RUN

A prepared program may be executed using the RUN statement. The program is executed starting at the lowest numbered statement. All variables and stacks are cleared (set to zero) before program execution starts.

The form of the RUN statement is:

*RUN

After program completion, BASIC prompts the user with an asterisk (*) in the left margin, indicating that it is ready for additional command statements. If the program should contain errors, an error message is printed that indicates the error and the line number containing the error, and program execution is terminated. Again, a prompt is given. The program must now be edited to correct the error and rerun. This process is continued until the program runs properly without producing any error messages. See "Errors" (Page 17-75) for a discussion of error messages.

Occasionally a program contains an error that causes it to enter an unending loop. In this case, the program never terminates. The user may regain control of the program by typing CONTROL-C (CNTRL-C). This aborts the program and returns control to the user. Storage is not altered in this process. CONTINUE resumes program execution. RUN clears the storage and restarts program execution.

SCRATCH

SCRATCH clears all current storage areas used by BASIC. This deletes any commands, programs, data, strings, or symbols currently stored by BASIC.

SCRATCH should be used for entering a new program from the terminal keyboard to ensure that old program lines are not mixed with new program lines. It also assures a clear symbol table. The form of the SCRATCH statement is:

```
*SCRATCH
```

Before destroying stored information, the user is asked "SURE?" A "Y" reply causes SCRATCH to proceed. Any other response cancels SCRATCH. For example:

```
*SCRATCH      (Scratch statement entered.)
SURE? Y       (Are you sure, answer Y (YES,))
*             (BASIC is ready for a new entry.)
```

VERIFY

The VERIFY statement permits you to check a file placed on mass storage without affecting the current program. The VERIFY command responds by indicating the name of the file found, and if the file is correct. A form of the VERIFY command is:

```
*VERIFY "name"
```

The string "name" can be the name of the record the user desires to verify or it may be a null (""); in which case, BASIC verifies the first record encountered. For example,

```
*VERIFY "STARTREK"  
FOUND STARTREK VER 1.0 03/11/77  
FILE OK  
*
```

In the above example, the file containing the Startrek dump is verified. Note, that the name of the file is printed immediately as soon as the label record is encountered. The FILE OK message is printed after the data record is read and verified. VERIFY performs a checksum on the contents of all data in the file. Using the VERIFY command does not destroy any program data in memory.

During a VERIFY, one of two error messages may be generated. They are:

```
SEQ ERR and  
CHKSUM ERR.
```

A sequence error (SEQ ERR) is generated if the file records are not in sequence. For example, if two consecutive label records are read an error is generated, as a BASIC file consists of a label file followed by a data file. The form of the sequence error is

```
SEQ ERR
```

Type a blank after the SEQ ERR message. This will clear the error. The entire file must be reread.

A checksum error (CHKSUM ERR) is generated if the computed CRC for the record in question does not match the checksum included in the record. The form of the CRC error message is

```
CHECKSUM ERR - IGNORE?
```

A Y in response to the question ignore aborts the error message and the record is considered valid. Do not ignore the checksum error unless there is no other way to recover the data. If a checksum error is flagged, the chances are very good that the data in the designated record is faulty.

The command VERIFY is not available if BASIC is patched to use an ASR console terminal as the load dump device. See "Product Installation" on Page 0-19 of the "Introduction" to this Software Reference Manual.

Statements Valid in the Command or Program Mode

The statements in this section may be used in either the command or the program mode. A few of them have only subtle uses in one mode or the other. Because they may be used in both modes, they are listed in this section.

CLEAR

CLEAR sets the contents of all variables, arrays, string buffers, and stacks to zero. The program itself is not affected. The command is generally used before a program is rerun to insure a fresh start if the program is started with a command other than RUN. The form of the CLEAR statement is:

```
*10 CLEAR          (BASIC)
10 CLEAR varname   (EXTENDED BASIC)
```

All variables, arrays, string buffers, etc., are cleared before program is executed by RUN. Therefore, a clear statement is not required. However, a program terminated prior to execution (by a STOP command or an error) does not set these variables, etc., to zero. They are left with the last value assigned. *If the variable name (varname) is specified, the CLEAR command clears the named variable, array, or DEF FN (user defined function).*

Note that the memory space used by string variables and arrays is not freed when CLEAR varname is used. String values should be set to null (for example, A\$ = "") before clearing so the string space can be recovered.

For example:

```
CLEAR A      Clears variable A
CLEAR A$     Clears the string variable A$
CLEAR A(     Clears the dimensioned variable A(
```

If a section of the program is to be rerun after appropriate editing, the variables, arrays, dimensions, etc., should be reinitialized. You can accomplish this by using the CLEAR statement in the command mode.

CNTRL (CONTROL)

CONTROL is a multi-purpose command used to set various options and flags. The form of the CONTROL statement is:

```
CNTRL iexp1, iexp2
```

The various CNTRL options are:

	iexp1	iexp2
CNTRL	0,	nnn
CNTRL	1,	n
CNTRL	2,	n
CNTRL	3,	n
CNTRL	4,	n

CNTRL 0

The CNTRL 0, nnn command sets up a GOSUB routine to process CONTROL-B characters. The line number of the routine is specified as "iexp2." When a CONTROL-B is entered from the terminal program, control is passed to the specified statement (beginning at the line iexp2) via a GOSUB linkage, after the statement being executed is completed. For example:

```
10 CNTRL 0,500
20 FOR A=1 TO 9
30 PRINT A,A*A,A*A*A
40 NEXT A
50 END
500 PRINT "THAT TICKLES"
510 RETURN
```

*RUN

1	1	1	
2	4	8	
3	9	27	(CONTROL-B typed)

THAT TICKLES			
4	16	64	(CONTROL-B typed)

THAT TICKLES			
5	25	125	
6	36	216	
7	49	343	
8	64	512	
9	81	729	

END AT LINE 50

*

During the execution of the program containing these three statements, a **CONTROL-B** from the keyboard momentarily interrupts the regular execution of the program. The program completes the line in progress and then enters the subroutine at line 500 printing the string

THAT TICKLES

It then moves to the next statement, a **RETURN**. This causes the program to continue with normal program execution. **NOTE:** The **CNTRL 0, nnn** must be executed before it is operational.

CNTRL 1

The **CNTRL 1, n** command sets the number of digits permitted before the exponential notation is used. Normal mode $M = 6$. For example:

CNTRL 1,2 (Numbers ≥ 100 are to be in exponential format.)

```
*PRINT 101
1.01000E+02
```

CNTRL 2

The **CNTRL 2, n** command controls the H8 front panel LED display mode. The control functions are:

CNTRL 2,0 Turn display off (Normal mode).

CNTRL 2,1 Turn display on without update. (For writing into a display. See the example under "The SEG Function, SEG (NARG)" on Page 17-65).

CNTRL 2,2 Turn display on with update (to monitor a register or memory location).

CNTRL 3

The **CNTRL 3, n** command controls the size of a print zone. This is normally 14. However, **CNTRL 3, n** can change the number of spaces in a print zone.

```
*
*CNTRL 3,5
*PRINT 1,2,3,4,3,2,1,0
      1   2   3   2   3   4   3   2   1   0
```

CNTRL 4 The **CNTRL 4,n** command turns the hardware clock on and off. **CNTRL 4,0** turns the clock off and **CNTRL 4,1** turns it on. Once the clock has been turned off, clock dependent functions such as **PAUSE iexp**, and **PAD(** cannot be used. Turning the clock off increases execution speed approximately 11%.

NOTE: The **CNTRL 1** through **CNTRL 4** commands permanently reconfigure loaded **BASIC**. A new program does not clear them to the original state. You can reset them to their original state by using the appropriate form of the Control statement in the Command mode.

DIM (DIMENSION)

The **DIMENSION** statement explicitly defines the maximum dimensions of array variables. A single dimension array is often called a vector. The form of the **DIMENSION** statement is:

```
*DIM varname (iexp1[, . . . . ., iexpn]) [, varname2 ( . . . . . )]
```

The expressions "iexp1" through "iexpn" are integer expressions specifying the bounds of each dimension. Dimensions are 0 to "expn." So, for example, the statement:

```
DIM A(5,5)
```

reserves an array 6×6 or 36 values. If the dimensioned variable is numeric, the values are preset to zero. If the dimensioned variable is a string, all the values are preset to a null string.

You may declare several variables in one **DIMENSION** statement by separating them with commas. For example:

```
*DIM A6(3,2), B(5,5), C3(10,10)
```

dimensions the following arrays

<u>VARIABLE</u>	<u>SIZE</u>	
A6	4 by 3	12 elements
B	6 by 6	30 elements
C3	11 by 11	121 elements

You can place a DIMENSION statement anywhere in a multiple statement line and it can appear anywhere in the program. However, an array can only be dimensioned once in a program unless it is cleared. DIMENSION statements must be executed before the first reference to the array, although good programming practices place all DIMENSION statements in a group among the first statements of a program. This allows them to be easily identified and changed if alterations are required later. The following example demonstrates the use of the DIMENSION statement with subscripted variables and a two-level FOR statement.

```
*LIST
10 REM DIMENSION DEMO PROGRAM
20 DIM A(5,10)
30 FOR B=0 TO 5
40 LET A(B,0)=B
50 FOR C=0 TO 10
60 LET A(0,C)=C
70 PRINT A(B,C);
80 NEXT C:PRINT :NEXT B
90 END
```

```
*RUN
0 1 2 3 4 5 6 7 8 9 10
1 0 0 0 0 0 0 0 0 0 0
2 0 0 0 0 0 0 0 0 0 0
3 0 0 0 0 0 0 0 0 0 0
4 0 0 0 0 0 0 0 0 0 0
5 0 0 0 0 0 0 0 0 0 0

END AT LINE 90
*
```

FOR AND NEXT

FOR and NEXT statements define the beginning and end of a program loop. A program loop is a set of repeated instructions. Each time they are repeated they modify a variable in some way until a predetermined condition is reached, causing the program to exit from the loop. The FOR NEXT statement is of the form:

```
FOR var = nexp1 to nexp2 [STEP nexp3]
NEXT VAR
```

When BASIC encounters the FOR statement, the expressions nexp1, nexp2 and nexp3 (if present) are evaluated. The variable “var” may be a scalar numeric variable, or it may be an element of a numeric array. It is assigned a value of “nexp1.” For example:

```
*FOR A=2 TO 20 STEP 2:PRINT A;:NEXT A
 2  4  6  8 10 12 14 16 18 20
```

causes the program to execute as long as A is less than or equal to 20. Each time the program passes through the loop, the variable A is incremented by 2 (the STEP number). Therefore, this loop is executed a total of 10 times. When incremented to 22, program control passes to the line following the associated NEXT statement. It is important to note that the initial value used for the variable is the value assigned to the variable expression when it entered the FOR-NEXT loop. For example:

```
*A=10:FOR A=2 TO 20 STEP 2:PRINT A;:NEXT A
 2  4  6  8 10 12 14 16 18 20
*
```

Prior to execution, the variable A is assigned the value 10. The program passes through the loop 10 times. A is reset to 2 and then increments from 2 to 20.

If “nexp2” \geq 0, and the initial value of var \geq “nexp2,” the loop terminates. For example, the program:

```
*LIST
10 FOR J=2 TO 18 STEP 4
20 J=18
30 PRINT J;:NEXT J
40 END

*RUN
18
END AT LINE 40
*
```

is only executed once, since the value of J = 18 is reached on the first pass, satisfying the termination condition.

A loop created by the statement:

```
*FOR A=20 TO 2 STEP 2:PRINT A;:NEXT A
20
*
```

is executed only once, as the initial value exceeds the terminal value. However, if this example is modified to read:

```
*FOR A=20 TO 2 STEP -2:PRINT A;:NEXT A
20 18 16 14 12 10 8 6 4 2
*
```

the negative step allows normal operation.

In summary, for positive STEP values, the loop is executed until the variable (var) is greater than the final assigned value (nexp2). For negative STEP values, the loop is executed until the variable (var) is less than the final assigned value (nexp2).

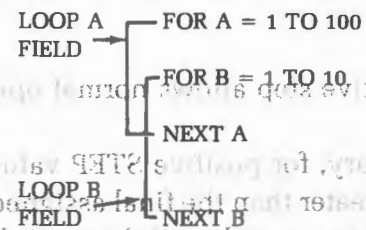
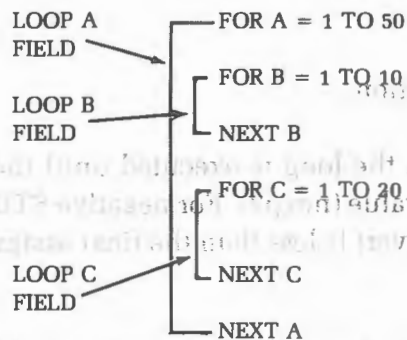
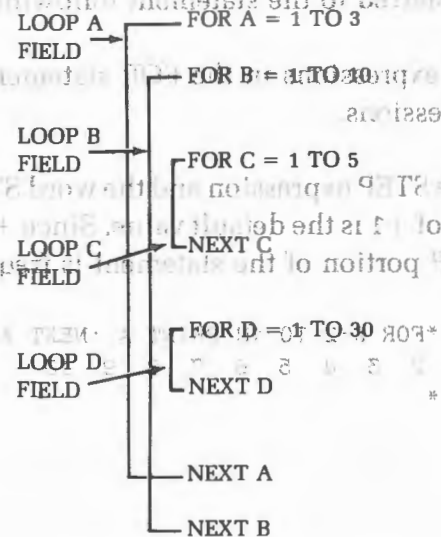
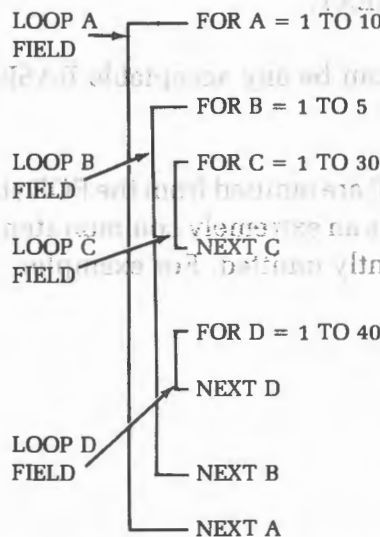
If the loop does not terminate, execution is transferred to the statement following the FOR statement. Therefore, a series of statements may be executed using the incremented value of the variable. If the loop does terminate, execution is transferred to the statement following NEXT.

The expressions in the FOR statement can be any acceptable BASIC numeric expressions.

If the STEP expression and the word STEP are omitted from the FOR statement, a step of +1 is the default value. Since +1 is an extremely common step value, the STEP portion of the statement is frequently omitted. For example:

```
*FOR A=2 TO 10:PRINT A;:NEXT A
2 3 4 5 6 7 8 9 10
*
```

Nesting is a technique frequently used in programming. It consists of placing one or more loops completely inside another loop. The field or operating range of the loop (the lines from the FOR statement to the corresponding NEXT statement) must not cross the field of another loop. The following two examples show legal and illegal nesting of FOR NEXT loops.

LEGAL NESTINGILLEGAL NESTINGTwo-Level NestingThree-Level Nesting

Note that both columns of nesting illustrations are shown in two-level and three-level forms. However, right-hand columns are not truly nesting but a crossover of FOR and NEXT loops (fields), and therefore are illegal. Also note that each of these examples uses the implied STEP value of 1.

The depth of nesting depends upon the amount of memory space available in Extended BASIC. BASIC limits FOR loops to 5 levels. Exceeding 5 levels generates an overflow error.

It is possible to exit from a FOR NEXT loop without reaching the variable termination value. This can be done using a conditional transfer such as an IF statement within the loop. However, control can only be transferred into a loop if the loop is left during prior program execution without being completed. This ensures the assignment of values to the termination and step variables.

Both FOR and NEXT statements can appear anywhere on a multiple statement line.

The NEXT statement does not require the variable. If the variable is not given, BASIC will NEXT the innermost FOR loop.

FREE (EXTENDED BASIC ONLY)

The FREE statement displays the amount of memory used by EX. B.H. BASIC and any program material. It also displays the total amount of free space left, which is dependant on the amount of memory in the computer and the program size. This command is particularly valuable when you are gauging the size of the program's data structure and establishing limits on a DIMENSION command. The FREE command also indicates the cause of memory overflow errors. The form of the FREE statement is:

*FREE

The form of the printout is:

TEXT = nnnn	(Bytes used by program text.)
SYMB = nnnn	(Bytes used by variables and arrays.)
FORL = nnnn	(Bytes used by FOR loops.)
GSUB = nnnn	(Bytes used by GOSUBs.)
STRN = nnnn	(Bytes used by STRING.)
WORK = nnnn	(Bytes used by expression and function evaluation.)
FREE = nnnn	(Total number of free bytes.)

For example, running the program

```
*10 GOSUB 10
```

BASIC soon returns a memory overflow error. Executing FREE shows the user a very large GOSUB table. This, and the statement provided in the error message, enables one to determine the program is in a GOSUB loop.

```
*FREE
TEXT = 9
SYMB = 0
FORL = 0
GSUB = 0
WORK = 0
STRN = 0
FREE = 7248
*10GOSUB 10
*RUN

! ERROR - MEM OVR AT LINE 10
*FREE
TEXT = 9
SYMB = 0
FORL = 0
GSUB = 7232
WORK = 0
STRN = 0
FREE = 16
*
```

GOSUB AND RETURN

A subroutine is a section of program performing some operation required one or more times during program execution. Complicated operations on a volume of data, mathematical evaluations too complex for user defined functions, or a previously written routine are all examples of processes best performed by a subroutine.

More than one subroutine is allowed in a single program. Good programming practices dictate that subroutines should be placed one after another at the end of the program in line number sequence. A useful practice is to assign distinctive line number groups to subroutines.

For example, a main program uses line numbers through 300. The 400 block is assigned to subroutine #1 and the 500 block is assigned to subroutine #2. Thus, any errors, program modifications, etc., involving the subroutine are easily identified.

Subroutines are normally placed at the end of a program, but before data statements if there are any.

Program execution begins and continues until a GOSUB statement is encountered. The form of the GOSUB statement is:

```
*GOSUB iexp
```

where iexp is the first line in the subroutine. Once GOSUB is executed, program control transfers to the first line of the subroutine and the subroutine is executed. For example:

```
60 GOSUB 500
```

in this example, control (the sequence of program execution) is transferred to line 500 in the program after line 60 is executed. The first line in the subroutine may often be a remark to identify the subroutine, or it may be any executable statement.

Once program control is transferred to a subroutine, program execution continues in the normal line-by-line manner until a RETURN statement is encountered. The RETURN statement is of the form:

```
RETURN
```

RETURN causes the program control to return to the statement **following** the original GOSUB statement. A subroutine must always be terminated by a RETURN.

Before BASIC transfers control to a subroutine, the next sequential statement to be processed after the GOSUB statement is internally recorded. The RETURN statement draws on this stored information to restart normal program execution. Using this technique, BASIC always knows where to transfer control, no matter how many times subroutines are called.

Subroutines can be nested in the same manner that FOR NEXT statements can be nested. That is, one subroutine can call another subroutine, and if necessary, that subroutine may call a third subroutine, etc. If, during execution of the subroutine a RETURN is encountered, control is returned to the line following the GOSUB calling the subroutine. Therefore, a subroutine can call another subroutine, even itself. Subroutines can be entered at any point and can have more than one RETURN. Multiple RETURN statements are often necessary when a subroutine contains conditional statements imbedded in it, which cause different subroutine completions dependent on the program data.

It is possible to transfer to the beginning or to any part of the subroutine. Multiple entry points and returns make the GOSUB statement an extremely versatile tool.

Up to 10 levels of GOSUB nesting are permitted in BASIC. *Extended BASIC permits unlimited GOSUB nesting. However, nesting uses memory and excessive nesting depth will cause an overflow.*

GOTO

The GOTO statement provides unconditional transfer of program execution to another line in the program. The GOTO statement is of the form:

***GOTO iexp**

When this statement is executed, program control transfers to the line number specified by the integer expression "iexp." For example:

```
10 LET A=1
20 GOTO 40
30 LET A=2
40 PRINT A
50 END
```

***RUN**

```
1
END AT LINE 50
```

*

Program lines in this example are executed in the following order:

10, 20, 40, 50

Line 30 is never executed because the GOTO statement in line 20 unconditionally transfers control to line 40. After the unconditional transfer takes place, normal sequential execution resumes at line 40.

IF THEN (IF GOTO)

The IF THEN (IF GOTO), conditionally transfers program execution from the normal consecutive order of program lines, depending on the results of a relation test. The forms of the IF statement are:

IF expression { THEN iexp or
GOTO s
IF expression THEN statement

The expression frequently consists of two variables combined by the relational operators described in "Relational Operators" (Page 17-18). In the first form, if the result of the expression is true, control passes to the specified line number (iexp). In the second form, control passes to the statement following THEN on the remainder of the line. If the result of the expression is false, control passes to the next line or to a statement separated from the IF THEN statement by a colon (:). The following examples show use of the IF THEN statement.

```
10 READ A
20 B=10
30 IF A=B THEN 50
40 PRINT "A< >B",A:END
50 PRINT "A=B",A
60 DATA 10,5,20
70 END
```

```

*CONTINUE
A < B 5
END AT LINE 40
*CONTINUE
A < B 5
END AT LINE 40

```

NOTE: The expression can be an arbitrarily complex expression. For example:

IF (A<3) AND NOT (B>C) THEN

LET

The LET statement assigns a value to a specific variable. The form of the LET statement is:

```
LET var = nexp,           or
LET var$ = sexp
```

The variable "var" may be a numeric variable or a string variable "var\$." The expression may be either an arithmetic "nexp" or a string expression "sexp" (*Extended BASIC*). However, all items in a statement must be either numeric or string, they cannot be mixed. If they are mixed, a type conflict error is flagged. NOTE: Unlike standard BASIC, multiple assignments are not permitted. For example,

```
LET A=B=3
```

causes A to be set to 65,535 (true) if B is equal to 3, or it causes A to be set to 0 (false) if B is not equal to 3. It does not cause both A and B to be set to 3.

You may omit the key word LET if you prefer. For example, the following two statements produce identical results.

```
10 LET A = 6
AND
10 A = 6
```

The LET statement is often referred to as an assignment statement. In this context, the meaning of the equal (=) symbol should be understood as it is used in BASIC. In ordinary algebra, the formula $Y = Y + 1$ is meaningless. However, in BASIC the equal sign denotes replacement rather than equality. Thus, the formula $Y = Y + 1$ is translated as add 1 to the current value of Y and store the new result at the location indicated by the variable Y.

Any values previously assigned to Y are combined with 1. An expression such as $D = B + C$ instructs BASIC to add the values assigned to the variables B and C and store the resultant value at the location indicated by the variable D. The variable D is not evaluated in terms of previously assigned values, but only in terms of B and C. Therefore, if previous assignments gave D a different value, the prior value is lost when this statement is executed.

LIST

This command lists the program on the console terminal for reviewing, editing, etc. The form of the list command is:

```
LIST [iexp]           B.H. BASIC
LIST [iexp1], [iexp2] EX. B.H. BASIC
```

Line numners are indicated by the optional integer expressions. If no line numbers are specified, the entire program is listed. *If a single line number ("iexp 1") is specified, EX. B.H. BASIC lists that single line.* BASIC lists the indicated line and the balance of the program lines. You can use a CONTROL-O or CONTROL-C to abort the listing *If both of the optional line numbers are specified, separated by a comma (,), all lines within the range of iexp 1 to iexp 2 are listed.* You can abort a listing by using the control characters. Refer to "Editing Commands" (Page 17-71) or to "Appendix B" (Page 17-83) for a complete explanation of these functions.

The following are examples of the LIST command.

```
10 LET A=5:LET B=6
20 PRINT A,B,A+B,
30 LET C=A/B
40 PRINT C
50 END
*RUN
5      6      11      .833333
```

```
END AT LINE 50
*LIST
```

```
10 LET A=5:LET B=6
20 PRINT A,B,A+B,
30 LET C=A/B
40 PRINT C
50 END
*LIST 20
```

```
20 PRINT A,B,A+B,
*LIST 20,40
```

```
20 PRINT A,B,A+B,
30 LET C=A/B
40 PRINT C
*
```

} EX. B.H. BASIC only

ON . . . GOSUB

The ON . . . GOSUB statement allows you to program a computed GOSUB. When you use the ON . . . GOSUB statement, use a RETURN at the end of the subroutine to return program control to the statement **following** the ON . . . GOSUB statement. The form of the ON . . . GOSUB statement is:

```
ON iexp1 GOSUB iexp2, . . . . ., iexpn
```

When it is processing an ON . . . GOSUB statement, BASIC evaluates the expression “iexp1” and uses the result as an index to the list of statement numbers iexp2 thru iexpn. If the expression “iexp1” evaluates to 1, for example, control is passed to the line number given by the expression “iexp2.” If the expression “iexp1” evaluates to 3, for example, control is passed to line number given by the expression “iexp4.” If the expression “iexp1” evaluates to 0, or to an index greater than the number of statement numbers listed, control is passed to the next program statement.

ON . . . GOTO

The ON . . . GOTO statement allows you to perform a computed GOTO. The form of the ON . . . GOTO statement is:

```
ON iexp1 GOTO iexp2, . . . . ., iexpn
```

When it is processing an ON . . . GOTO statement, BASIC evaluates the expression “iexp1” and uses the result as an index to the list of statement numbers iexp2 thru iexpn. If the expression “iexp1” evaluates to 1, for example, control is passed to the line number given by the expression “iexp2.” If the expression “iexp1” evaluates to 3, for example, control is passed to line number given by the expression “iexp4.” If the expression “iexp1” evaluates to 0, or to an index greater than the number of statement numbers listed, control is passed to the next program statement.

OUT

The OUT statement is used to output binary numbers to an output port. The form of the OUT statement is:

```
OUT iexp1, iexp2
```


The expression "iexp1" is used as the port address, and "iexp2" is the value to be placed at that port. Both iexp1 and iexp2 are decimal numbers. The low-order 8-bits generated by the decimal numbers in iexp1 or iexp2 are used. If you wish to write iexp1 and iexp2 in octal notation for ease in conversion to the actual binary values, write a subroutine or function to perform octal to decimal conversion.

PAUSE

The PAUSE statement causes BASIC to delay before executing the next statement. There are two forms of PAUSE. In BASIC the form of the PAUSE statement is:

PAUSE

Once the PAUSE statement is executed, no further statements are executed until you type a console terminal character. You can terminate PAUSE by typing any key, and this will not cause the character to be echoed, but it is good practice to consistently use one character such as space to terminate PAUSE. In Extended BASIC the form of the PAUSE statement is:

PAUSE [iexp]

You can also terminate PAUSE by specifying an optional time duration with iexp. If iexp is specified, PAUSE waits 2 times iexp milliseconds. After 2 times iexp milliseconds pass, normal program execution continues.

The PAUSE statement is particularly useful when you are viewing long outputs on a CRT display. You can insert a PAUSE at appropriate points in the program, allowing you to view the information on the CRT before continuing execution.

POKE

The POKE statement is used to write values into an assigned H8 memory location. The form of the POKE statement is:

POKE iexp1, iexp2

The low-order 8-bits of iexp2 are inserted into memory location iexp1. NOTE: iexp1 and iexp2 must be given as decimal numbers. If you wish to use octal numbers for ease in referencing to binary notation, you must use a separate octal to decimal subroutine or function to generate these numbers.

CAUTION: You can damage BASIC when using the POKE statement, causing a failure which could result in loss of program material and/or require reloading BASIC itself. The POKE statement should be confined to areas of memory, not used by the BASIC interpreter.

PORT

The PORT statement is used to direct the result of a print statement to an I/O port other than the console terminal port (372₈ or 250₁₀). You can also use it to direct the console terminal operations to another port. One of the primary uses of the PORT statement is directing data printing to a printer. The form of the PORT statement is:

```
PORT iexp
```

The expression iexp is the desired port number (in decimal). If iexp is positive, the print function is transferred to the indicated port number while the statement (in the command mode) or the program (in the program mode) is being executed. After execution is complete, the printing function returns to the console terminal. If iexp is negative, the keyboard functions of the console terminal are transferred to the desired port, along with the printing functions.

The console terminal is then permanently transferred to the selected port and an additional PORT instruction must be issued at this new console terminal to return operation to port 250. Recovery from accidental assignment to a nonexistent port is accomplished by an RST0 followed by a warm start (PC = 040 103).

PRINT

The PRINT statement is used to output **data** to the console terminal. The form of the PRINT statement is:

```
PRINT [nexp1 sep1 . . . nexpn(sepn)]
```

The expressions and separators contained within the brackets are optional. When used without these optional expressions and separators, the simple PRINT statement outputs a blank line followed by a carriage-return line feed.

Printing Variables

The PRINT statement can be used to evaluate expressions and to simultaneously print their results, or to simply print the results of a previously evaluated

expression or evaluations. Any expression contained in the PRINT statement is evaluated before the result is printed. For example:

```
10 A=4:B=6:C=5+A
20 PRINT
30 PRINT A+B+C
40 END
*RUN
```

```
19
```

```
END AT LINE 40
*
```

All numbers are printed with a preceding and following blank. You can use PRINT statements anywhere in a multiple statement line. NOTE: The terminal performs a carriage-return line feed at the end of each PRINT statement unless you use the separators described in "Use of the , and ;" (Page 17-52). Thus, in the previous example, the first PRINT statement outputs a carriage-return line feed and the second print statement outputs the number 19 followed by a carriage-return line feed.

Printing Strings

The PRINT statement can be used to print a message (a string of characters). The string may be alone or it may be used together with the evaluation and printing of a numeric value. Characters to be printed are designated by enclosing them in quotation marks ("). For example:

```
10 PRINT "THIS IS A HEATH H8"
*RUN
THIS IS A HEATH H8

END AT LINE 65535
*
```

The string contained in a PRINT statement may be used to document the variable being printed. For example:

```
10 LET A=5:LET B=10
20 PRINT "A + B",A+B
30 END
*RUN
A + B          15

END AT LINE 30
*
```

When a character string is printed, only the characters between the quotes appear. No leading or trailing blanks are added as they are when a numeric value is printed. Leading and trailing blanks can be added within the quotation marks.

Use of the , And ;

The console terminal is normally initialized with 80 columns divided into five zones. (See CNTRL 3, n for exception.) Each zone, therefore, consists of 14 spaces. When an expression in the PRINT statement is followed by a comma (,) the next value to be printed appears in the next available print zone. For example:

```
10 A=5.55555:B=2
20 PRINT A,B,A+B,A*B,A-B,B-A
30 END
*RUN
5.55554          2          7.55554          11.1111          3.55554
-3.55554

END AT LINE 30
*
```

NOTE: The sixth element in the PRINT list is the first entry on a new line, as the five print zones of a 72-character line were used.

Using two commas together in a PRINT statement causes a print zone to be skipped. For example:

```
10 A=5.55555:B=2
20 PRINT @,B,A+B,,A*B,A-B,B-A
30 END
*RUN
5.55554          2          7.55554          11.1111
2.55554          -3.55554

END AT LINE 30
*
```

If the last expression in a PRINT statement is followed by a comma, no carriage-return line feed is given when the last variable is printed. The next value printed (by a later PRINT statement) appears in the next available print zone. For example:

```

10 LET A=1:LET B=2:LET C=3
20 PRINT A,
30 PRINT B
40 PRINT C
50 END
*RUN
1           2
3

END AT LINE 50
*
```

At certain times, it is desirable to use more than the designated five print zones. If such tighter packing of the numeric values is desired, a semicolon (;) is inserted in place of the comma. A semicolon does not move the next output to the next PRINT zone, but simply prints the next variable, including its leading and trailing blank. For example:

```

10 LET A=1:LET B=2:LET C=3
20 PRINT A;B;C
30 PRINT A+1;B+1
40 PRINT C+1
50 END
*RUN
1  2  3
2  3
4

END AT LINE 50
*
```

NOTE: If either a comma or a semicolon is the final character in a PRINT statement, no final carriage-return line feed is printed.

READ AND DATA

The READ and DATA statements are used in conjunction with each other to enter data into an executing program. One statement is never used without the other. The form of the statements are:

```
READ var1, . . . , varn
DATA exp1, . . . , expn
```

The READ statement assigns the values listed in the DATA statement to the specified variables var1 through varn. The items in the variable list may be simple variable names, arrays, or *string variable names*. Each one is separated by a comma. For example:

```
5 DIM A (2,3)
10 READ C,B$,A (1,2)
20 DATA 12,"THIS IS SIX",56
30 PRINT C,B$,A (1,2)
*RUN
12          THIS IS SIX  56

END AT LINE 65535
*
```

Because data must be read before it can be used in the program, READ statements generally occur in the beginning of a program. You may, however, place a READ statement anywhere in a multiple statement line. The type of expression in the DATA statement must match the type of corresponding variable in the READ statement. When the DATA statement is exhausted, BASIC finds the next sequential DATA statement in the program. NOTE: BASIC does not automatically go to the next DATA statement for every READ statement. Therefore, one DATA statement may supply values for several READ statements if DATA statement contains more expressions than the READ statement has variables.

DATA statements may contain arbitrarily complex expressions to represent the data values. Each value expression is separated from other value expressions by a comma. A field in the DATA statement may be left null by means of two adjacent commas. This causes the associated variable to retain its old value. For example:

```
10 A=1:B=1:C=1
20 READ A,B,C
30 PRINT A,B,C
40 DATA 3,,4
50 END
*RUN
3          1          4

END AT LINE 50
*
```

If a DATA statement appears on a line, it must be the only statement on the line. DATA statements may not follow any other statement on the line. Other statements should not follow DATA statements.

DATA statements do not have to be executed to be used. That is, they may be the last statement in a program, and be used by a READ statement executed earlier in the program. However, if DATA statements appear in a program in such a place that they are executed (there are executable statements beyond the DATA statement), the executed DATA statement has no effect. Therefore, location of DATA statements is arbitrary as long as the expressions contained within the DATA statements appear in the correct order. However, good programming practice dictates all DATA statements occur near the end of the program. This makes it easy for the programmer to modify the DATA statements when necessary.

If an expression contained in a DATA statement is bad, the illegal character error message is printed. All subsequent READ statements also cause the message. If there is no data available in the data table for the READ statement to use, the no data error message is printed.

If the number of expressions in the data list exceed those required by the program READ statements, they are ignored, and thus not used.

REM (REMARK)

The REMARK statement lets you insert notes, messages, and other useful information within your program in such a form that it is not executed. The contents of the REMARK statement may give such information as the name and purpose of the program, how the program may be used, how certain portions of the program work, etc.. Although the REMARK statement inserts comments into the program without affecting execution, they do use memory which may be needed in exceptionally long programs.

REMARK statements must be preceded by a line number when used in the program. They may be used anywhere in a multiple statement line. The message itself can contain any printing character on the keyboard and can include blanks. BASIC ignores anything on a line following the letters REM.

RESTORE

The RESTORE statement causes the program to reuse data starting at the first DATA statement. It resets the DATA statement pointer to the beginning of the program. The RESTORE statement is of the form:

RESTORE

For example:

```
10 READ A,B,C
20 PRINT A,B,C
30 RESTORE
40 READ D,E,F
50 PRINT D,E,F
60 DATA 1,2,3,4,5,6,7,8
70 END
*RUN
1          2          3
1          2          3

END AT LINE 70
*
```

This program does not utilize the last five elements of the DATA statement. The RESTORE command resets the DATA statement pointer and the READ D,E,F, statement uses the first three data elements, as does the initial READ statement.

The CLEAR command includes the RESTORE function.

STEP

The STEP command permits you to step through a program a single line or a few lines at a time. The form of the step command is:

```
STEP iexp
```

where the integer expression iexp indicates the number of lines to be executed before stopping. Execution of the desired lines is indicated by the prompt NXT = nnnn, where nnnn is the next line number to be executed. A STEP 2 is required to execute the first program line. All future single-line executions require a STEP or STEP 1. For example:

```

10 READ A,B,C
20 PRINT A,B,C
30 RESTORE
40 READ D,E,F
50 PRINT D,E,F
60 DATA 1,2,3,4,5,6,7,8
70 END

*CLEAR

*STEP 3
1      2      3
NXT= 30
*STEP
NXT= 40
*STEP
NXT= 50
*STEP
1      2      3
NXT= 60
*STEP 2

END AT LINE 70
*
```

Program Mode Statements

PROGRAM MODE statements are valid only when utilized within a program. If they are entered in the command mode, an illegal use error is flagged.

DATA

The DATA statement discussed in “Read and Data” (Page 17-54) is a program only statement, although it is used in conjunction with a READ statement, which may be used in either the command or program modes.

DEF FN

The DEF FN statement defines single line program functions created by the user. The form of the DEF FN statement is:

```
DEF FN varname (arg1 [,arg2, . . . . ,argn] ) = expr
```

The variable name (varname) must be a legal string or numeric variable name and cannot be previously dimensioned. However, it may be previously defined. The latest definition takes precedence. The argument list “(arg1 [arg2, ,arg3])” must be supplied to indicate a function. NOTE: The arguments are real, not dummy variables, and do change as evaluation proceeds.

```
10 REM DEFINE A SQUARE FUNCTION
20 DEF FN S1(I) = I * I
30 PRINT FN S1(3),I,FN S1(5),I
40 END

*RUN
9          3          25          5

END AT LINE 40
*
```

END

The END statement causes control to return to the command mode. An END statement message is typed, giving the line number of the END statement. END also causes the next statement pointer to be set to the beginning of the program so a CONTINUE resumes execution at the beginning of the program.

An END statement may appear anywhere in the program, as many times as desired. If a program does not contain an END statement, it “runs off the end.” In this case, BASIC generates a pseudo end statement at line 65,535.

INPUT AND LINE INPUT

The INPUT statement is used when data is to be READ from the terminal keyboard or from a mass storage device **working through the console terminal**. The form of the INPUT statement is:

```
INPUT prompt;var1, . . . , varn
```

If the first element in the list following the INPUT statement is a string, INPUT assumes it is a PROMPT and types the string in place of a question mark (?). *If no prompt string is desired but the first variable is a string variable, a leading semicolon is inserted. For example:*

```
INPUT ;S3$(2)
```

This statement tells BASIC that the data to come from the console terminal is to be placed in a dimensioned string named S3.

Data input from the console terminal has a format identical to the DATA statement.

NOTE: Responses to string inputs must be enclosed in quotes.

Expressions may be supplied and null fields cause the variable to retain its previous value. If the user response does not supply sufficient data to complete the INPUT statement, another “?” prompt is issued, requesting more data input at the terminal. **CAUTION:** If you supply too much data, it will be ignored. The next INPUT statement issues a fresh READ to the terminal.

The response to the LINE INPUT statement cannot be continued on another line, as they are terminated by the return key.

When there are several values to be entered via the input statement, it is helpful to print a message explaining the data needed, using the prompt string. For example:

```
10 INPUT "THE TIME IS";T
```

When this line of the program is executed, BASIC prints

```
THE TIME IS
```

and then waits for a response.

The LINE INPUT statement is used to input one line of string data from the console terminal and assign it to a string variable. Its form is identical to the INPUT form, but the string should not be enclosed in quotes.

STOP

The STOP statement causes BASIC to enter the command mode. The message stating the line number of the STOP is printed. The next line pointer is left after the STOP statement, so a CONTINUE statement causes execution to resume on the line immediately after the STOP statement. The STOP statement is of the form:

```
STOP
```

The STOP statement can occur several times throughout a single program with conditional jumps determining the actual end of the program. The following example uses the STOP statement to examine a variable during execution.

```

10 A=1:B=2:C=3
20 PRINT A,B,C
30 END

*RUN
1           2           3

END AT LINE 30
*15STOP

*RUN

STOP AT LINE 15
*PRINT A
1
*15                               Stop deleted

*RUN
1           2           3

END AT LINE 30
*
```

PREDEFINED FUNCTIONS

Introduction

There are 26 predefined functions in EX. B.H. BASIC and 16 predefined functions in B.H. BASIC. These functions perform standard mathematical operations such as square roots, logarithms, string manipulation, and special features. Each function has an abbreviated three- or four-letter name, followed by an argument in parentheses. As these functions are predefined, they may be used throughout a program when required. Predefined functions use numeric expressions (nexp), integer expressions (iexp), and in EX. B.H. BASIC, string expressions (sexp). Function key words are automatically followed by an open parenthesis "(".

The abbreviation (narg) is used to indicate a numeric argument, a decimal number lying in the approximate range of 10^{-38} to 10^{+37} . Certain functions do not permit the argument to assume this wide range, as indicated in the function description.

The predefined functions may be used in either the command or program mode.

Arithmetic and Special Feature Functions

The Arithmetic and Special Feature Functions supported by BASIC are identical for EX. B.H. BASIC and B.H. BASIC with the exception of the functions Maximum, Minimum, Space, Tab, and Tangent. These are discussed later in this section. (See Page 17-67.)

THE ABSOLUTE VALUE FUNCTION, ABS (nexp)

The ABSOLUTE VALUE Function gives the absolute value of the argument. The absolute value is the positive portion of the numeric expression. For example:

```
*PRINT ABS(-5.5)
5.5                or,

*PRINT ABS(SIN(3.5))
.350783
*
```

NOTE: The sine of 3.5 radians is $-.350783$.

THE ARC TANGENT FUNCTION, ATN (nexp)

The ARC TANGENT Function returns the arc tangent of the argument. For example:

```
*PRINT ATN(1/1)*57.296;"DEGREES"
45.0001 DEGREES
*PRINT 4*ATN(1)
3.14159
*
```

NOTE: $\pi = 3.14159$

THE COSINE FUNCTION, COS (nexp)

The COSINE function returns the COSINE of the argument (nexp) expressed in radians. For example:

```
*PRINT COS(60/57.296)
.500003
*
```

THE EXPONENTIAL FUNCTION EXP (NEXP)

The EXPONENTIAL function returns the value e^{nexp} . If “nexp” exceeds 88, an overflow error is flagged, as the result exceeds 10^{38} . If “nexp” is less than -88 , an overflow error occurs. An example of the exponential function is:

```
*PRINT EXP(1),EXP(2),EXP(COS(60/57.296))
2.71828          7.38905          1.64873
*
```

THE INTEGER FUNCTION, INT (narg)

The INTEGER function returns the value of the greatest integer value, not greater than “narg.” If the argument is a negative number, the INTEGER function returns the negative number with the same or smaller absolute value. For example:

```
*PRINT INT (38.55)
38

*PRINT INT (-3.3)
-3
```

THE LOGARITHM FUNCTION, LOG (nexp)

The LOGARITHM function returns the natural logarithm (LOG to the base e) of the argument. You can find the Logarithms of a number N in any other base by using the formula:

$$\text{LOG}_a N = \text{LOG}_e N / \text{LOG}_e a$$

where “a” represents the desired base. Most frequently, “a” is 10 when you are converting to common logarithms. For example:

```
*PRINT "A POWER RATIO OF 2 IS";10*(LOG(2)/LOG(10));"DECIBELS"
  A POWER RATIO OF 2 IS 3.0103 DECIBELS
*
```

THE PAD FUNCTION, PAD (0)

The PAD function returns the value of the keypad pressed on the H8 front panel. For example:

```
*PRINT PAD(0)
  6                      The #6 key was pressed.
```

The PAD function uses all the front panel debounce and repeat software contained in PAM-8. (See “The Segment Function,” Page 17-65, for an additional example.)

NOTE: The PAD function must be completely executed before any other function will respond. Therefore, CONTROL-C, etc., will not work until you press an H8 front panel key.

THE PEEK FUNCTION, PEEK (iexp)

The PEEK function returns the numeric value of the byte at memory location iexp.

THE PIN FUNCTION, PIN (iexp)

The PIN function returns the value input from port “iexp” where iexp is a decimal expression ranging from 0-255. For example:

```
*A=PIN(38)
```

Where “A” now contains the data that was at port #38 (46 octal).

THE POSITION FUNCTION, POS (0)

The POSITION function returns the current terminal printhead (cursor) position. Although the numeric argument (0) is ignored, it must be present to complete the function. The value returned is a decimal number indicating the column number of the printhead (cursor) position. For example:

```
*PRINT POS(0), POS(0), POS(0); POS(0); POS(0)
  1          15          29   33   37
*
```

THE RANDOM FUNCTION, RND (narg)

The RANDOM number function returns the next element in a pseudo random series. The RANDOM number generator is not truly random, and may be manipulated by controlling the argument. If narg>0, the random number generator

returns the next random number in the series. If $\text{narg} = 0$, the random number generator returns the previously returned random number. If $\text{narg} < 0$, the value "narg" is used as a new seed for a random number, thus starting an entire new series. Using these three inputs to the random number series, the programmer may continuously return the same number while de-bugging the program, determine what the series of numbers will be when the program is run, or start a series of new random numbers each time BASIC is loaded. For example:

```
10 FOR A=0 TO 2
20 PRINT RND(1)
30 NEXT
40 END
```

```
*RUN
.93677
.566681
.53128
```

```
END AT LINE 40
```

```
*RUN
.564484
.787262
.332306
```

```
END AT LINE 40
```

```
*20PRINT RND(0)
```

```
*RUN
.332306
.332306
.332306
```

```
END AT LINE 40
```

```
*20PRINT RND(-1)
```

```
*RUN
6.25305E-02
6.25305E-02
6.25305E-02
```

```
END AT LINE 40
```

```
*20PRINT RND(-5)
```

```
*RUN
.460968
.460968
.460968
```

```
END AT LINE 40
```

```
*
```


THE SEGMENT FUNCTION, SEG (narg)

The SEG function returns a numeric value which is the correct 8-bit binary number to display the digit on the H8 front panel LED's. The argument must be an integer between 0 and 9. The following program demonstrates the use of PAD, POKE, and SEG in EX. B.H. BASIC. See the second example for a B.H. BASIC program.

```

10 REM A PROGRAM TO USE THE FRONT PANEL LEDS. CNTRL 2,1 TURNS
20 REM ON THE LEDS WITHOUT UPDATE. THE KEYPAD NOW DRIVES THE
30 REM DISPLAY THRU BASIC. 8203 IS THE FIRST LED MEM LOCATION.
40 CNTRL 2,1
50 A=8203
60 FOR I=A TO A+8
70 POKE I,SEG(PAD(0))
80 NEXT I
90 GOTO 60
*RUN

```

When this program is executed, the H8 front panel LEDs respond to the H8 keypad numeric entries. To escape from the program you would type CONTROL-C and then press a key on the H8 front panel. NOTE: If BASIC is to be used, the EX. B.H. BASIC command CNTRL cannot be used to turn on the displays and turn off the update. Therefore, the program is modified to use a POKE command at the first line which stops display update. The program escape routine is still the same. When you are using BASIC, the program is:

```

40 POKE 8200,2
50 A=8203
60 FOR I=A TO A+8
70 POKE I,SEG (PAD(0))
80 NEXT I
90 GOTO 60

*RUN

*

```

THE SIGN FUNCTION, SGN (narg)

The SIGN function returns the value +1 if “narg” is a positive value, 0 if “narg” is 0, and -1 if “narg” is negative. For example:

```
*PRINT SGN(5.6)
1

*PRINT SGN(-500)
-1

*PRINT SGN(12-12)
0

*
```

THE SINE FUNCTION SIN (nexp)

The SIN function returns the sine of the argument (nexp) expressed in radians. For example:

```
*PRINT SIN(30/57.296)
.499999

*
```

SQUARE ROOT FUNCTION, SQR (narg)

The SQUARE ROOT function returns the square root of “narg.” The argument “narg” must be greater than or equal to 0 (for example, positive).

```
*FOR A=0 TO 5:PRINT A,SQR(A),A*A:NEXT
0      0      0
1      . 1      1
2      1.41421  4
3      1.73205  9
4      2      16
5      2.23607  25

*
```

USER DEFINED FUNCTION, USR (narg)

This function calls a user-supplied machine language function. The user-supplied function should be stored in high memory above BASICs user set high memory limit. You should place the starting address of your machine language function at location USRFCN. (See “Appendix C.”) A RET instruction exits from the user-defined function. An example of the USR function is given in “Appendix E,” Page 17-111.

THE FREE SPACE FUNCTION, FRE (0) [B.H. BASIC ONLY]

The FREE function returns the number of free bytes of program and variable storage area available. You must supply the dummy argument 0 to complete the function. This function is not available in EX. B.H. BASIC. Most frequently, this function is used in the command mode in conjunction with a PRINT statement. For example:

```
*PRINT FRE (0)
1224
*
```

THE MAXIMUM FUNCTION, MAX (nexp1, . . . ,nexpn)

The MAXIMUM function returns the maximum value of all the expressions which are arguments of the function. For example:

```
10 LET A=1
20 PRINT MAX(COS(A),SIN(A)/COS(A))
30 END
*RUN
1.55741
END AT LINE 30
*
```

The expression containing the maximum value is the expression for the tangent of 1 radian, (1.55741).

THE MINIMUM FUNCTION, MIN (nexp1, . . . ,nexpn)

The MINIMUM function returns the lowest value of all the expressions contained in the argument. For example:

```
*PRINT MIN(1,2,3,4,.5)
.5
*
```

THE TANGENT FUNCTION, TAN (nexp)

The TANGENT Function returns the TANGENT of the argument "nexp" expressed in radians. For example:

```
*PRINT TAN (45/57.296)
.999996
*
```

THE SPACE FUNCTION, SPC (iexp)

The SPACE function spaces the printhead (cursor) iexp spaces to the right of its present position. For example:

```
*PRINT 12,14,SPC(20);600
12          14          600
*
```

THE TAB FUNCTION TAB (iexp)

The TAB function moves the printhead (cursor) to the iexp th column. NOTE: If the printhead is at or past the iexp th column, the function is ignored. For example:

```
*PRINT TAB(20);60,70
*          60      70
```

String Functions (Extended BASIC Only)

Extended BASIC contains various functions for processing character strings in addition to the functions used for mathematical operations. These functions allow the program to concatenate two strings, access a part of string, generate a character string corresponding to a given number, or generate a number for a given string.

THE CHARACTER FUNCTION, CHR\$ (iexp)

The CHARACTER function returns a string that consists of a single character. The character generated has the ASCII code "iexp." NOTE: "iexp" is a decimal number and must be converted to octal for comparison with most ASCII character tables. See "Appendix D" on Page 0-52 of the "Introduction." For example:

```
*PRINT CHR$(65)
A
*PRINT CHR$(70)
F
*
```

NOTE: If iexp = 0, the generated string is null.

THE STRING FUNCTIONS, STR\$ (narg)

The *STRING* function encodes the argument (narg) into ASCII in the same format used by the *PRINT* statement for numbers. These characters are returned as a string, with leading and trailing blanks. For example:

```
*PRINT STR$(100)      } STR$ function
10C                   }
*PRINT "100"          } Normal string printing
10C                   }
*
```

THE ASCII FUNCTION, ASC (sexp)

The *ASCII* function returns the ASCII code for the first character in the string expression (sexp). If the string is a null, the *ASCII* function returns a zero. The return is a decimal number and must be converted to octal for comparison to most ASCII tables. See "Appendix D" on Page 0-52 of the "Introduction." For example:

```
*PRINT ASC("ABC")
65
*PRINT CHR$(65)
A
*
```

THE LEFT STRING FUNCTION, LEFT\$ (sexp, iexp)

The *LEFT STRING* function returns the "iexp" left-most characters of the string expression (sexp). If "iexp" equals 0, the null string is returned. For example:

```
*PRINT LEFT$("HELLO, THIS IS A TEST",10)
HELLO, THI
*
```

THE RIGHT STRING FUNCTION, RIGHT\$ (sexp, iexp)

The *RIGHT STRING* function returns the right-most "iexp" characters of the string expression (sexp). If "iexp" equals 0, the null string is returned. For example:

```
*PRINT RIGHT$("HELLO, THIS IS A TEST",10)
IS A TEST
*
```

THE MIDDLE STRING FUNCTION, MID\$ (sexp, iexp [, iexp2])

The MIDDLE STRING function returns the right-hand substring of the string expression "sexp" starting with the "iexp1" th character from the left-hand side (the first character is 1). The return continues for "iexp2" characters or to the end of the string if the optional terminating expression "iexp2" is omitted. For example:

```
*PRINT MID$("HELLO, THIS IS A TEST",10,10)
IS IS A TE
*
```

THE NUMERIC VALUE FUNCTION, VAL (sexp)

The NUMERIC VALUE function returns the numeric value of the number encoded in the string expression (sexp). For example:

```
*PRINT VAL (".003E-1")
3.00000E-04
*
```

THE LEN FUNCTION, LEN (sexp)

The LEN function returns the length of the string expression "SEXP." For example:

```
*PRINT LEN("HOW LONG IS THE STRING?")
23
```

EDITING COMMANDS

BENTON HARBOR BASIC provides several commands used to halt program execution, erase characters, delete lines, add lines, and provide other editing functions. A great number of these editing commands are common to all the Heath H8 Software packages. Their operation is covered in detail in the “Console Driver” section (Page 0-36) of the “Introduction” to this Software Reference Manual.

Control-C, CNTRL-C

CONTROL-C is a general-purpose cancel key. It can be used to stop a mass storage input or output operation, stop program execution, stop a listing, and to stop a program during an input statement. Using CONTROL-C results in the CC error message (CNTRL-C in EXTENDED BASIC). NOTE: A CONTROL-C causes the program to terminate at the end of a current statement.

You can continue program execution by using the CONTINUE statement.

Inputting Control

The following control characters take effect when you are inputting information from the control terminal.

BACKSPACE, BKSP/CNTRL-H

The BACKSPACE key (or a CONTROL-H) causes a one-character backspace. The backspace code is echoed to the terminal so devices with backspace capability physically backspace. Attempting to backspace into column zero is illegal and causes a terminal bell code to be echoed. NOTE: Backspace can be changed at configuration. See “Product Installation” on Page 0-19 of the “Introduction” to this Software Reference Manual.

RUBOUT

The RUBOUT key causes BASIC to discard the current line being inputted. A carriage-return feed line is sent to the console terminal, and the user may now re-enter the entire line. NOTE: Rubout can be changed at configuration. See “Product Installation” on Page 0-19 of the “Introduction” to this Software Reference Manual.

Outputting Control

The following control characters take effect only when you are outputting information to the console terminal. They should not be used when you are inputting information via the console terminal, as they affect characters being echoed to the console.

OUTPUT SUSPENSION AND RESTORATION, CNTRL-S AND CNTRL-Q

Type CONTROL-S to suspend the output and to suspend program execution. This command is particularly useful when you are using a video terminal; you can use the CONTROL-S (suspend) feature each time a screen is nearly filled and information at the top will be lost due to scrolling.

By typing CONTROL-Q, you permit BASIC to continue execution and outputting information to the terminal. CONTROL-Q cancels the CONTROL-S function.

The DISCARD FLAG, CNTRL-O and CNTRL-P

Type a CONTROL-O to toggle the DISCARD FLAG. Setting the DISCARD FLAG stops output on the terminal but does not halt program execution until you retype CONTROL-O or until you type a CONTROL-P to clear the DISCARD FLAG. CONTROL-O is often used to discard the remainder of long listings and other similar outputs. BASIC clears the discard flag when it returns to the command mode or when an INPUT statement is executed, so that the prompt will appear.

Command Completion

When you are inputting information from the console terminal in the command mode, or in response to an INPUT command, BASIC checks the incoming characters for the initial characters of a keyword. As soon as enough characters of a keyword are entered to uniquely identify it, and it is distinguished from a variable name, BASIC completes the keyword into the terminal. For example, to enter the command SCRATCH, type SC. Since SC uniquely determines SCRATCH and SC is not a legal variable name, BASIC types the characters RATCH immediately following the SC. Striking the backspace key backspaces over the entire word SCRATCH.

Function keywords are automatically followed by an open parenthesis “(”. Other keywords are immediately followed by a blank.

Enforced Lexical Rules

BASIC enforces two lexical rules during input.

1. Two adjacent alphabetical characters must start a keyword. For example, XX is illegal as no keyword starts with "XX" and "XX" is an illegal variable name. This rule is excepted when following a REMARK statement or when the characters are contained within quotes (indicating a string).
2. A quoted string must be closed; every quote character must have a mate on the same line.

Should a character be typed which is in violation of these rules, a bell code is echoed and the character is ignored.

General Text Rules

BLANKS

BASIC programs are generally "free format." That is, blanks (spaces) may be included freely with the following restrictions.

1. Variable names, keywords, and numeric constants may not contain imbedded blanks.
2. Blanks may not appear before a statement number.

LINE INSERTION

You can insert lines into a BASIC program by simply typing an appropriate line number followed by the desired line of text. This is done in response to the command mode prompt (an asterisk). Except when running a program, BASIC remains in the command mode, showing a single asterisk (*) as a prompt. NOTE: The text should immediately follow the last digit of the line number. Although intervening blanks are allowable, they waste memory. BASIC automatically inserts a blank when listing the text. For example:

```
*100PRINT "HEATH BASIC"  
LIST  
100 PRINT "HEATH BASIC"
```

LINE LENGTH

A line in BENTON HARBOR BASIC is restricted to 80 characters, and lines in Extended BENTON HARBOR BASIC are restricted to 100 characters. This restriction on line length is completely independent of the console width, which was established by the software configuration procedure (see Page 0-19 of the "Introduction" to this Software Reference Manual). NOTE: If the console terminal, for example, was set at a width of 34 characters, the console will display three complete lines for one line of BASIC.

LINE REPLACEMENT

Replace existing program lines by simply typing the line number and the new text. This is the same process you use to insert a new line. The old line is completely lost once the new line is entered.

LINE DELETION

Delete lines by typing the line number immediately followed by a carriage-return. You can leave blank lines by typing the single space before typing the carriage-return.

ERRORS

BASIC detects many different error conditions. When an error is detected, a message of the form:

```
! ERROR-(ERROR MESSAGE) [at line NNNNN]
```

is typed. BASIC returns to the command mode (if it is not already in the command mode), ringing the console terminal bell. If BASIC is in the command mode, the “at line NNNN” portion of the error message is omitted. For example:

```
*PRINT 1/0
! ERROR - /0
*10PRINT 1/0
*RUN

! ERROR - /0 AT LINE 10
*
```

NOTE: If a line of BASIC contains an error, you can correct it by retyping the entire line. Once the line number is typed, the contents of the old line are lost. To delete a line, type the line number and follow it with a carriage-return.

Error Messages

The following “Basic Error Table” describes the error messages generated by BENTON HARBOR BASIC and Extended BENTON HARBOR BASIC. Two columns of error messages are given, as BENTON HARBOR BASIC provides a shortened form of the error messages. An explanation is given after each error message in the Comments column.

Recovering from Errors

When an error is detected, BASIC enters the command mode with the variables and stacks as they were at the time of the error. Thus, the user can use PRINT and LET statements to examine and alter variable contents. Likewise, a GOTO statement can be used to set the next statement pointer to any desired statement number. Often, a combination of these techniques allows the user to continue a program with the error corrected.

NOTE: If program text in an EX. B.H. BASIC program is modified in any way, the GOSUB and FOR stacks are purged. If an error occurred in a GOSUB routine, or a FOR LOOP, the entire program must be restarted. If you modify text in a B.H. BASIC program, B.H. BASIC CLEARs all variables.

BASIC ERROR TABLE

BASIC	EXTENDED BASIC	COMMENTS
AC	<i>ARG CT</i>	Argument count. Incorrect number of arguments supplied to a DEF defined function.
CC	<i>CNTRL-C</i>	CONTROL-C. Program execution or other operation aborted by a CONTROL-C typed at the console terminal.
DE	<i>NO DAT</i>	Data exhausted. A READ called for more data than is available in the DATA statement.
/0	<i>/0</i>	Divide by zero. An attempt to divide by zero. NOTE: Either dividing by the number zero or dividing by an expression which evaluates to zero generates this error.
IN	<i>NUM</i>	Illegal number. The line number referenced by a command or program statement is not used by BASIC.
IU	<i>USE</i>	Illegal usage. A command or statement is used in the improper context.
NX	<i>NXT</i>	Next variable missing. No FOR statement matching the accompanying NEXT statement.
OV	<i>OVRFL</i>	Overflow. Memory space is filled by program text.
RE	<i>RTN</i>	Return error. A RETURN is encountered without a calling statement.
S#	<i>STAT#</i>	Statement number. The referenced statement number does not exist in the program.

BASIC	EXTENDED BASIC	COMMENTS
SY	<i>SYN</i>	Syntax error. Command, statement, or function uses incorrect separators, functions, etc..
MO	<i>MEM OV</i>	Table overflow. One of the internal tables has grown too large for memory. Check GOSUBs, FOR loops, and DIM.
SR	<i>SUBS RANG</i>	Subscript range. The subscript size of a dimensioned variable exceeded the size defined by the DIM statement.
SC	<i>SUBS CNT</i>	Subscript count. The number of subscripts assigned to a variable exceeds the number defined in the DIM statement.
ND	<i>DIM?</i>	Not dimensioned. The subscripted variable has not been dimensioned in a DIM statement.
IC	<i>ILL CHA</i>	Illegal character. An improper character is assigned to a command or function NOTE: In B.H. BASIC, an attempt to assign a string to a numeric variable results in an illegal character message.
FU	<i>FN ?</i>	Function error. No single line function defined by a DEF statement was found when the FN function was encountered. NOTE: The DEF FN must be executed prior to executing the FN function.
TE	<i>TAPE</i>	Tape error. An error in handling the mass storage device at the load dump port.

BASIC	EXTENDED BASIC	COMMENTS
	CNTRL-B	<i>CONTROL-B error. A CONTROL-B entered from the console terminal, but no CONTROL-B processing in the program.</i>
	STR LE	<i>String length error. The length of a string exceeded 255 characters.</i>
	TYP CNFLC	<i>Type conflict. String data supplied for a numeric variable or numeric data supplied for a string variable.</i>

APPENDIX A

Loading Procedures

Loading From the Software Distribution Tape

1. Load the tape in the reader.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. A single beep indicates a successful load.
5. Press GO on the H8 front panel.
6. The console terminal will respond with:

```
HEATH/WINTEK H8 BASIC
BENTON HARBOR BASIC ISSUE # 05.01.00
COPYRIGHT 01/77 BY WINTEK CORP.
```

7. Configure B.H. BASIC or EX. B.H. BASIC as desired, answering the following questions. Prompt each question by typing its first character on the console terminal keyboard.

```
•AUTO NEW-LINE (Y/N)?
•BKSP = 00008/
•CONSOLE LENGTH = 00080/
•HIGH MEMORY = 16383/
•LOWER CASE (Y/N)?
•PAD = 4/
•RUBOUT = 00127/
•SAVE?
•
```

8. Before executing SAVE, have the tape transport ready at the DUMP port.
9. To use BASIC directly from the distribution tape, type the return key at any time rather than a question prompt key. The Console Terminal will display:

```
B.H. BASIC # 05.01.00
```

```
*
```

BASIC is ready to use.



Loading From a Configured Tape

1. Load the tape in the tape transport.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. A single beep indicates a successful load.
5. Press GO on the H8 front panel.
6. The console terminal responds with:

B.H. BASIC # 05.00.00

*

BASIC is ready to use in the configured form.

APPENDIX B

A Summary of BASIC

For additional details, refer to the page number that is given with each of the following topics.

See Page

Numeric Data

17-9

Numbers may be real or integer with the following characteristics:

Range 10^{-38} to 10^{+37} .
Accuracy 6.9 digits.
Decimal range 0.1 to 999999.
Exponential format $(\pm) X.XXXXXXE (\pm) NN$.

Boolean Data

17-10

Integer numbers from 0 to 65535 represent two byte binary data from 00000000 00000000 to 11111111 11111111. Fractional parts of numbers between 0 and 65535 are discarded.

String Data (Extended BASIC Only)

17-10

Data is all printed in ASCII characters plus the BELL, BLANK and LINE FEED, with the following characteristics:

Maximum string length 255 characters.
Enclosure Quotation marks (") on both ends.
Multiple lines Not allowed for a single string.

Variables

17-11

Variables are named by a single letter (A through Z), or a single letter followed by a single number (0 through 9). For example: A or A6.

See Page

Subscripted Variables

17-12

Subscripted variables are named like variables, but are followed by dimensions in parentheses. Subscripted variables are of the form:

$A_{n(N_1, N_2, \dots, N_r)}$ For example: $A(1, 2, 7)$ or $A_6(1, 5)$.

You must use a DIMENSION statement to define the range and number of allowable subscripts for a variable.

Arithmetic Operators

17-14

Listed in order of priority. Operators on the same line have equal precedence. Parenthetical operations are performed first. Precedence is left to right if all other factors are equal.

<u>SYMBOL</u>	<u>EXPLANATION</u>
—	Unary negation logical compliment
↑	Exponentiation. Ex BASIC only
* /	Multiplication division
+ —	Addition subtraction

Relational Operators

17-18

<u>SYMBOL</u>	<u>EXPLANATION</u>
=	Equal to
<	Less than
< =	Less than or equal to
>	Greater than
> =	Greater than or equal to
< >	Not equal to

See Page

Boolean Operators

17-19

Boolean operators perform the Boolean (logical) operations on two integer operands. The operands must evaluate to integers in the range of 0 to 65535. The operators are:

NOT	Logical complement, bit by bit
OR	Logical OR, bit by bit
AND	Logical AND, bit by bit

String Variables

17-21

String variables may be either subscripted or nonsubscripted. They take the same form as numeric or Boolean variables but are followed by a dollar sign (\$) to indicate a string variable. For example: A\$ A6\$ A\$(1,2,7) or A6\$(1,5).

String Operators

17-22

String expressions may be operated on by the relational operators as well as the plus (+) symbol. The plus symbol is used to perform string concatenation.

Line Numbers

17-25

When it is used in the program mode, BASIC requires that each line be preceded by an integer line number in the range 1 to 65535.

The Command Mode

17-23

The command mode does not use line numbers. Statements are executed when a carriage-return is typed.

Multiple Statements on One Line

17-25★

BASIC permits multiple statements on one line. Each statement is separated from the others by a colon (:). DATA statements may not appear on lines with other statements.

★See "Basic Statements."

Command Mode Statements

<u>COMMAND</u>	<u>FORM</u>	<u>DESCRIPTION</u>	<u>SEE Pg.</u>
<i>BUILD</i>	<i>BUILD iexp1, iexp2</i>	<i>Automatically generates program line numbers starting at iexp1 in steps of iexp2.</i>	17-27
CONTINUE	CONTINUE	Resumes program execution.	17-27
<i>DELETE</i>	<i>DELETE [iexp1, iexp2]</i>	<i>Deletes program lines between iexp1 and iexp2.</i>	17-28
DUMP	DUMP "name"	Saves current program "name" on mass storage media at load dump port; "name" is up to 80 ASCII characters.	17-28
LOAD	LOAD "name"	Loads program "name" from mass storage media at load dump port; "name" is up to 80 ASCII characters. Current program is destroyed.	17-29
RUN	RUN	Start execution of current program. Preclears all variables, stacks, etc..	17-30
SCRATCH	SCRATCH SURE?Y	Clears all program and data storage area. Any response to SURE but Y cancels SCRATCH.	17-31
VERIFY	VERIFY "name"	Performs a checksum on the mass storage record titled "name." No response if record is bad.	17-31

Command and Program Mode Statements

<u>COMMAND</u>	<u>FORM</u>	<u>DESCRIPTION</u>	<u>SEE Pg.</u>
CLEAR	CLEAR [<i>varname</i>]	Clears all variables, arrays, string buffers, etc. <i>Optionally clears named variable (varname).</i> <i>Specifies functions and arrays as V(.</i>	17-33
CONTROL	CNTRL <i>iexp1</i> , <i>iexp2</i>	CNTRL 0 sets a GOSUB to line <i>iexp2</i> when a CONTROL-B is typed.	17-34
		CNTRL 1 sets <i>iexp2</i> digits before exponential format is used.	17-35
		CNTRL 2 controls the H8 front panel. If <i>iexp2</i> : = 0, display off; = 1, display on without update; = 2, display on with update.	17-35
		CNTRL 3 sets the width of a print zone to <i>iexp2</i> columns.	17-35
		CNTRL 4 controls the H8 hardware clock. <i>iexp2</i> = 0, clock off. <i>iexp2</i> = 1, clock on.	17-36
DIMENSION	DIM <i>varname</i> (<i>iexp1</i> [, . . . , <i>iexpn</i>]) [, <i>varname2</i> (. . .)]		
		Defines the maximum size of variable arrays.	17-36
FOR/NEXT	FOR <i>var</i> = <i>nexp1</i> TO <i>nexp2</i> [STEP <i>nexp3</i>]		
	NEXT <i>var</i>	Defines a program loop. <i>Var</i> is initially set to <i>nexp1</i> . Loop cycles until NEXT is executed; then <i>var</i> is incremented by <i>nexp3</i> (default is +1). Looping continues until <i>var</i> > <i>nexp2</i> (or less than <i>nexp2</i> if STEP is negative). The statement after NEXT is then executed.	17-37

<u>COMMAND</u>	<u>FORM</u>	<u>DESCRIPTION</u>	<u>SEE Pg.</u>
<i>FREE</i>	<i>FREE</i>	<i>Displays the amount of memory assigned to tables and text</i>	17-41
GOSUB/ RETURN	GOSUB iexp RETURN	Transfers execution sequence of program to line iexp (the beginning of a subroutine). RETURN returns execution sequence to the statement following the calling GOSUB.	17-42
GOTO	GOTO iexp	Unconditionally transfers the program execution sequence to the line iexp.	17-44
IF/THEN	IF expression THEN iexp IF expression THEN statement	If the expression is true, control passes to iexp line or to "statement." If the relation is false, control passes to the next independent statement.	17-45
LET	LET var = nexp <i>LET var\$ = sexp</i>	Assigns the value nexp (<i>or sexp in the case of strings</i>) to the variable var (<i>or var\$</i>). LET keyword is optional.	17-46
LIST	LIST[iexp1] [,iexp2]	Lists the entire program on the console terminal. Lists the line iexp1 or the range of lines iexp1 to iexp2.	17-47
ON/GOSUB	ON iexp1 GOSUB iexp2, . . . ,iexpn.	Permits a computed GOSUB. Iexp1 is evaluated and acts as an index to line numbers iexp2 thru iexpn, each pointing to a different subroutine.	17-48

<u>COMMAND</u>	<u>FORM</u>	<u>DESCRIPTION</u>	<u>SEE Pg.</u>
ON/GOTO	ON iexp1 GOTO iexp2, . . . ,iexpn	Permits a computed GOTO. iexp1 is evaluated and acts as an index to line numbers iexp2 thru iexpn.	17-48
OUT	OUT iexp1, iexp2	Outputs a number iexp2 to output port iexp1.	17-48
PAUSE	PAUSE (<i>iexp</i>)	Ceases program execution until a console terminal key is typed. <i>Ceases program execution for 2 X iexp mS.</i>	17-49
POKE	POKE iexp1, iexp2	Writes a number iexp2 into memory location iexp1.	17-49
PORT	PORT <i>iexp</i>	<i>Assigns the print function to port iexp. Assigns the Console Terminal function to the device at port iexp if iexp is negative.</i>	17-50
PRINT	PRINT(<i>nexp. sep1 . . . nexpn(sepn)</i>)	<i>Prints the value of the expression (s) exp with a leading and trailing space. Expressions may be numeric or string. If the separator is a comma, the next print zone is used. If the separator is a semicolon, no print zones are used. No separator prints each expression value on a new line.</i>	17-50
READ & DATA	READ var1, . . ,varn DATA exp1 . . ,expn	The READ statement assigns the values exp1 thru expn in the data to the variables var1 thru varn.	17-54

<u>COMMAND</u>	<u>FORM</u>	<u>DESCRIPTION</u>	<u>SEE Pg.</u>
REMARK	REM	Text following the REM is not executed and is used for commentary only.	17-55
RESTORE	RESTORE	Causes the program to reset the DATA pointer, thus reusing data at the first DATA statement.	17-56
STEP	STEP iexp	Executes iexp lines of the program. Then returns BASIC to the command mode.	17-57

Program Mode Statements

DEF	DEF FN varname (arg list) = exp		
		Defines a single-line program function created by the user.	17-58
END	END	Causes control to return to the command mode.	17-58
INPUT	INPUT prompt; var1, . . . ,varn		
		Reads data from the console terminal. <i>String data must be enclosed in quotes.</i>	17-59
LINE INPUT	LINE INPUT prompt; var1, . . . ,varn		
		<i>Reads string data from the terminal. String data for LINE INPUT is not enclosed in quotes.</i>	17-59
STOP	STOP	Causes BASIC to enter the command mode when the statement containing STOP is executed.	17-60

Predefined Functions

<u>FUNCTION</u>	<u>DEFINITION</u>	<u>SEE Pg.</u>
ABS (nexp)	Returns the absolute value of nexp.	17-61
ATN (nexp)	Returns the arctangent of nexp (radians).	17-62
COS (nexp)	Returns the cosine of nexp (radians).	17-62
EXP (nexp)	Returns e^{nexp} .	17-62
INT (narg)	Returns the integer value of narg.	17-62
LOG (nexp)	Returns the natural logarithm of nexp.	17-62
PAD (0)	Returns the value of the H8 front panel key pressed. Includes key debounce.	17-63
PEEK (iexp)	Returns the numeric value at memory location iexp.	17-63
PIN (iexp)	Returns the data input from port iexp.	17-63
POS (0)	Returns the current terminal printhead (cursor) position (by column number).	17-63
RND (narg)	Returns a random number. If narg > 0, RND is next in the series. If narg = 0, RND is the previous random number. If narg < 0, RND algorithm uses narg as a new seed.	17-63
SEG (narg)	Returns the correct eight-bit number to display narg (0-9) on the H8 LEDs.	17-65
SGN (narg)	Returns +1 if narg is positive. Returns -1 if narg is negative. Returns 0 if narg is zero.	17-66
SIN (nexp)	Returns the sine of nexp (radians).	17-66
SPC (iexp)	Positions printhead (cursor) iexp columns to the right.	17-68
SQR (narg)	Returns the square root of narg.	17-66

<u>FUNCTION</u>	<u>DEFINITION</u>	<u>SEE Pg.</u>
TAB (iexp)	Position printhead (cursor) to the iexp th column.	17-68
USR (narg)	Calls a user-written machine language function to evaluate narg.	17-66
FRE (0)	Returns the amount of free memory in B.H. BASIC.	17-67
MAX (nexp1, . . . ,nexpn)	Returns the maximum value of expressions nexp1 thru nexpn.	17-67
MIN (nexp1, . . . ,nexpn)	Returns the minimum value of expressions nexp1 thru nexpn.	17-67
TAN (nexp)	Returns the tangent of nexp (radians).	17-67
CHR\$ (iexp)	Returns the ASCII character iexp.	17-68
STR\$ (narg)	Returns narg encoded into ASCII with leading and trailing blanks as in the print statement.	17-69
ASC (sexp)	Returns the ASCII code for the first character in the string sexp.	17-69
LEFT\$ (sexp, iexp)	Returns the left iexp characters of the string sexp.	17-69
RIGHT\$ (sexp, iexp)	Returns the right iexp characters of the string sexp.	17-69
MID\$ (sexp, iexp1) [,iexp2]	Returns the substring of the string sexp starting with the iexp1 th character and ending with the iexp2 th character if iexp2 is specified. If not specified, returns iexp1 th character to the end.	17-70
VAL (sexp)	Returns the numeric value of the number encoded in the string.	17-70
LEN (sexp)	Returns the length of sexp.	17-70

Editing Commands

<u>COMMAND</u>	<u>FUNCTION</u>	<u>SEE Pg.</u>
CONTROL-C	General-purpose cancel. Returns BASIC to monitor mode from any operation or program execution.	17-71
CONTROL-S	Suspends the output to the console terminal and suspends program execution.	17-72
CONTROL-Q	Restores the output to the console terminal and restores program execution.	17-72
CONTROL-O	Toggles the output discard flag. Does not stop program execution.	17-72
CONTROL-P	Clears the discard flag set by CONTROL-O.	17-72

